Optimized automation system for compact engine powerplants

David Andersson

Master's thesis
Supervisors: Rami Berg, Wärtsilä Finland Oy
Jerker Björkqvist, Åbo Akademi University
Jari Böling, Åbo Akademi University
Examinator: Prof. Margareta Björklund-Sänkiaho
Energy Technology, Vasa
Study programme in Chemical Engineering
Faculty of Science and Engineering
Åbo Akademi University
November 21, 2019

ABSTRACT

A common method used to monitor and control a complete process in today's industry is to use a Supervisory Control and Data Acquisition (SCADA) configuration. SCADA consists of server racks in server cabinets. In a centralized control room, there are several computers connected to SCADA. Process controllers manage local processes of a powerplant and report back to the SCADA system. This configuration has been used for a long time and has proven to be a robust solution. The disadvantage of this configuration is that it is relatively expensive. If this configuration is utilized for an industry at smaller scale, it might not be the most cost-effective alternative.

The objective of this thesis was to evaluate creating a HTML5-based user interface that would mimic the currently utilized SCADA system. Instead of having the centralized SCADA system, the HTML5-based user interface is distributed throughout the powerplant on Programmable Logic Controllers (PLC). Any device with a HTML5 supported web browser, connected in the same network as the PLC, can access the HTML5 user interface on the PLC.

To create the HTML5-based user interface, B&R's software Automation Studio with the mappView application was used. Various functionalities used in current SCADA were created in mappView. To store historical data, a third-party database management system was used on Windows 10. For redundancy, the database was configured to be replicated to another device.

Two touch panels were ordered to be evaluated for operating the created HTML5 user interface. The first tested operator panel was found to be insufficient for interacting with the HTML5 user interface, while the performance of the second tested panel PC indicated that it would be suitable to operate the created HTML5 user interface. The performance of the implemented database on the panel PC passed the requirements of a database to be implemented in an industrial setting.

Further investigation of the HTML5-based user interface would be to develop a fully-fledged user interface to test all functions of the current version of the SCADA in a single HTML5 user interface, and subsequently to evaluate if the panel PC still is able to perform without any major complications.

Key words: HTML5, user interface

ABSTRAKT

En vanlig metod för att övervaka en komplett process i dagens industri är en Supervisory Control and Data Acquisition (SCADA)-konfiguration, som består av modulära serverhyllor i ett flertal serverskåp. I den centraliserade SCADA-konfigurationen finns användargränssnitt samt historiska data, som nås av ett flertal datorer i ett kontrollrum. Programmerbara logiska styrenheter i olika sektioner av t.ex. ett kraftverk, hanterar dess lokala processer och för informationen vidare till SCADA-systemet. Denna konfiguration har visat sig vara en robust och välfungerande lösning. Nackdelen med denna konfiguration är att den är relativt dyr. Om serverkonfigurationen implementeras i en mindreskalig industri är det eventuellt inte den mest kostnadseffektiva lösningen.

Målet med denna avhandling var att utvärdera användning av ett HTML5-baserat användargränssnitt vars funktionalitet ska vara likt det nuvarande SCADA-systemet. Istället för det centraliserade SCADA-systemet finns det HTML5-baserade användargränssnittet i flera sektioner av kraftverket på ett flertal programmerbara logiska styrenheter (PLC). Enheter som har en webbläsare med HTML5-stöd och som är anslutna i samma nätverk som PLC:n kan nå HTML5-användargränssnittet.

För att skapa HTML5-användargränssnittet användes B & R:s programvara Automation Studio med applikationen mappView. Diverse funktionaliteter som finns i nuvarande SCADA-system återskapades i mappView. För att lagra historiska data implementerades en tredje parts databashanteringssystem på en Windows 10-installation. För redundans konfigurerades databasen till att bli replikerad till en annan enhet.

Två pekskärmar beställdes för att utvärdera hur väl HTML5-användargränssnittet kan styras. Den första, operatörspanelen visade sig inte vara lämplig för användargränssnittet, medan prestationen hos den andra, panel-PC:n, indikerade att den vore lämplig att använda. Det visade sig även att panel-PC:s prestanda var tillräcklig för att databasen kan implementeras i en industriell miljö.

I fortsättningen kunde man utveckla det HTML5-baserade användargränssnittet med ett fullt fungerande användargränssnitt, för att testa all funktionalitet hos det nuvarande SCADA-systemet i ett enda HTML5-användargränssnitt och därefter utvärdera om panel-PC:n fortfarande klarar av att prestera utan några större komplikationer.

Nyckelord: HTML5, användargränssnitt

TABLE OF CONTENTS

PREFACE

This thesis was conducted for Wärtsilä Finland Oy, Energy Business at Runsor, Vaasa during 2019. From Wärtsilä, I would like to thank Rami Berg for giving me the opportunity to conduct this thesis and for giving guidance throughout the work on this thesis. I would further like to thank everyone who has been involved with the advancements of the work of this thesis.

From Åbo Akademi University, I would like to thank my two supervisors Jerker Björkqvist and Jari Böling for their input of subjects to research and write about that is relatable to the thesis topic. I would also like to thank Professor Margareta Björklund-Sänkiaho at Åbo Akademi University, for the general guidance of the thesis procedures.

LIST OF ABBREVIATIONS

B&R	An automation company named after the two founders Erwin Bernecker & Josef Rainer
CSS	Cascading Style Sheets
DBMS	Database Management System
GPOS	General-Purpose Operating System
GUI	Graphical User Interface
HTML5	Hypertext Markup Language Version 5
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IEC	International Electrotechnical Commission
IIoT	Industrial Internet of Things
NoSQL	None Structured Query Language, often referring to a none relational database
OPC	Open Platform Communications
OPC-UA	Open Platform Communications - Unified Architecture
OPC-UA TSN	Open Platform Communications - Unified Architecture with Time-Sensitive Networking
PLC	Programmable Logic Controller
SCADA	Supervisory Control and Data Acquisition
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol / Internet Protocol
WAL	Write-Ahead Log
WOIS	Wärtsilä Operator Interface System

1 INTRODUCTION

The supervisory control systems in traditional powerplants are commonly based on a server model. This consists of a central unit in form of a server cabinet that is implemented to manage user interfaces as well as storage of historical data. This method is very robust and scalable to many user interface connections, as well as managing process values from many different units in a powerplant. This topology is called Supervisory Control and Data Acquisition (SCADA). Throughout a powerplant there are several Programmable Logic Controllers (PLCs) that monitor process data. These PLCs are subsequently connected in a network which can be monitored and controlled by the SCADA system (Daneels & Salter, 1999).

However, when implemented for smaller powerplants, the cost of the server setup will be a significant proportion of the total cost of the powerplant. To reduce costs for these smaller powerplants, a simpler setup is preferred, without compromising on the performance and functionality of the current SCADA system.

Industries worldwide are currently is in the beginning of the fourth industrial revolution (Bloem et al., 2014), where one key concept is modularizing the process automation (Wassilew et al., 2016). One solution to reduce costs is to decentralize the servers and have the user interfaces distributed in several modules throughout the powerplant.

Lately in automation, vendors have started offering hardware that communicate over open standards, as opposed to the past when it was common for each vendor to develop own communication standards. This makes it possible for customers to implement their solutions without being restricted to utilize vendor-specific hardware. Examples of these open standards are switching over to the field bus protocol OPC-UA and utilizing user interfaces based on HTML5-graphics.

In this thesis, the modular HTML5-based user interface solution is evaluated for containerized powerplants. The containerized powerplant comprises from one to

several internal combustion engines with an associated generator, mutually known as a generator set, or in short genset. In conjunction with the gensets, an optional common monitoring module is usually implemented as well.

In Wärtsilä's current powerplant standard, the Wärtsilä Operator Interface System (WOIS) is utilized as a control system, which is based on a server model (i.e. a SCADA system). Each genset has a process controller that translates process data from the engine and forwards the information to WOIS over the network.

In the modular solution, a light version of WOIS based on HTML5 is stored in each genset on the process controller in form of a webserver. Historical data is stored locally as well, but on the contrary to the user interface, the historical data is on a device with a general-purpose operating system (GPOS). Since the user interfaces are based on HTML5, they are accessed with HTML5 compatible hardware that are connected in the same network. Thus, the user interface implemented on each genset can be remotely accessed from any device connected to the same network.

1.1 Delimitations

The work conducted in this thesis was to create a proof of concept of a light version of WOIS. Individual functions of the current WOIS were evaluated in the new implemented light WOIS. If the decision is made to utilize the light WOIS for a customer's powerplant, the experience from this thesis can be implemented to develop a fully functional light WOIS.

2 POWERPLANT AUTOMATION BACKGROUND

In this chapter, various theoretical aspects of smaller powerplants and cohering functionalities are described.

2.1 OPC-UA

Traditionally in the automation industry, a common hierarchy model has been based on a figurative automation pyramid containing 5 levels, which can be seen in Figure 1. From the bottom level up there are sensors/actuators, Programmable Logic Controllers (PLCs), supervisory control and data acquisition (SCADA), manufacturing execution system (MES), and enterprise resource planning (ERP). The problem in this model has been the complexity of the communication between these levels. On the 2 lowest levels, the operational technology (OT) comprises a variety of communication protocols depending on the hardware from different vendors. On the remaining 3 top levels, commonly known as information technology (IT), devices communicate over the Internet Protocol. A vendor independent integration of OT and IT was historically achieved with the Open Platform Communications (OPC) protocol (Bruckner et al., 2019).

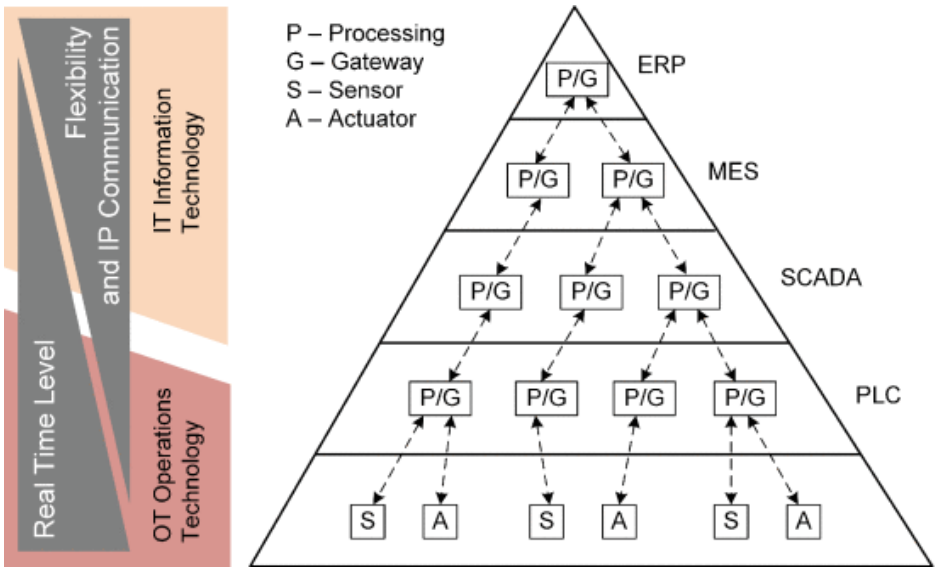

Figure 1 Hierarchy of the traditional automation pyramid consisting of 5 levels (Bruckner et al., 2019).

The Open Platform Communications, also called OPC Classic, is a set of standards that was first introduced in 1996 to the automation industry, to communicate between devices, regardless of vendor. There are three OPC standards: real-time Data Acquisition (DA), Historical Data Access (HDA), and Alarms & Events (A&E). The communication for these standards is based on Microsoft's Distributed Component Object Model (DCOM). Due to being based on DCOM, a Windows operating system must be used, which limit the selection of devices.

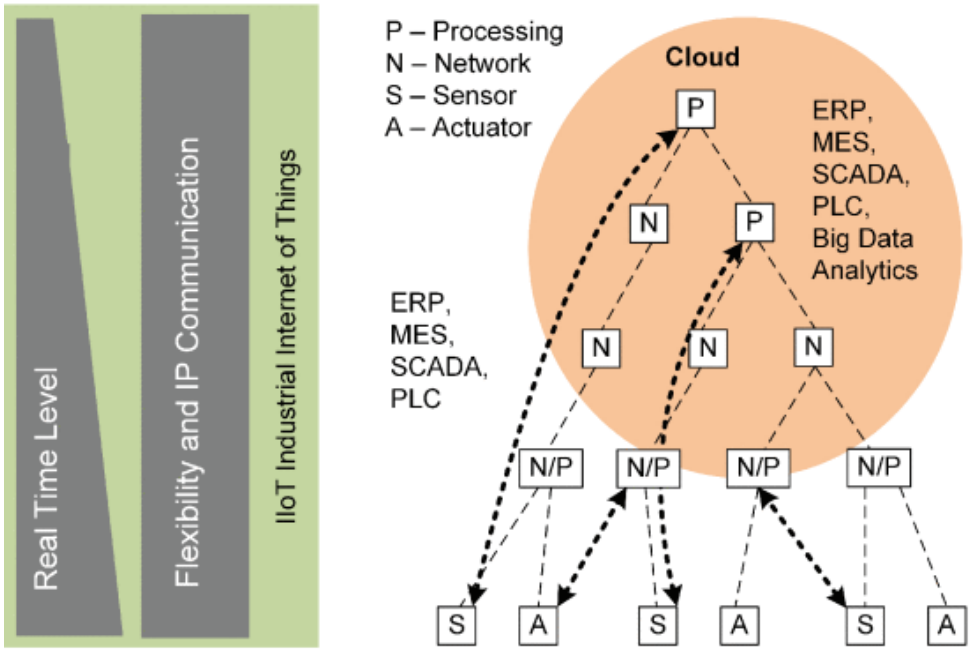

Figure 2 With a communication protocol, such as OPC-UA, all devices are interoperable with any other device. (Bruckner et al., 2019).

To advance on the platform-independency, the extension Unified Architecture (UA, as in OPC-UA) was introduced in 2006. OPC-UA is built upon the functions in the classic OPC, thus being partly compatible with devices using the classic OPC. Contrary to the classic OPC, the Unified Architecture can be scalable down to embedded devices with limited hardware resources, as well as a wide range of application domains. In addition to the gained platform-independency, OPC-UA is more secure, reliable, utilizes a more complex information modeling for metadata, and is dynamically discoverable by other OPC-UA servers and clients (Mahnke & Leitner, 2009). With the introduction of Unified Architecture, communication in the

aforementioned automation pyramid became interoperable between devices on all 5 levels, as illustrated in Figure 2. This further enables the possibility of Industrial Internet of Things (IIoT) within industry 4.0, where small devices, such as embedded devices, can post data to cloud storage.

OPC-UA communicates by sending various request and response messages between a server and a client. The server has a set of services that is provided to clients. Of the services described in the OPC-UA specifications, compiled by the OPC foundation, each vendor decides which of the services should be implemented in the vendors devices. These services are also called facets. The server is configured to what services clients can dynamically discover and access. One device may have one to several clients and/or servers. Each client can communicate with several servers and vice versa. If two servers are to communicate, one of the two servers will act as a client.

Alternatively, a PubSub model is used, where messages with an assigned topic are published from a device. Subsequently, other devices can subscribe to messages with certain topics. This method was introduced in addition to the server/client method, to increase real-time capability of messages. In this mode, no two devices are directly connected to each other. Instead, a Message Oriented Middleware manages the messages between publishers and subscribers. The publishers send their messages without knowing who the recipients are, or even if there are any recipients at all. Similarly, subscribers subscribe to certain topics without knowing who the publishers are. The Message Oriented Middleware is either a software or a hardware in a broker-based or a broker-less form. In the broker-less form, messages are routed in datagrams, such as User Datagram Protocol Multicast, in a network infrastructure. This is well-suited for a production environment with frequent transmission of small amounts of data. In the broker-based form, the broker (i.e. Message Oriented Middleware) communicates with publishers and subscribers with protocols such as Message Queuing Telemetry Transport (MQTT) or Advanced Message Queuing Protocol (AMQP). The broker-based form is more suitable if data is to be cloud integrated. Since PubSub as well as the server client method both are based on the same OPC-UA information model, they can easily co-exist in either a client or a server. In most cases,

publishers are servers and clients are subscribers.

The structure of an OPC-UA server is illustrated in Figure 3. The actual process data, also called real objects, are stored as attributes in nodes within an address space. In addition to the attributes in the nodes, there are references to other nodes, making the address space an interrelated network of nodes. The address space is structured hierarchically and represented in a tree to make it interoperable by both the server and clients. The information that is available to clients is stored in nodes within the address space. The address space can be further subset into views, which define what nodes the client can access (i.e. restricting unwanted nodes to the clients). Monitored items are created by clients, expressing interest in value changes of nodes in the server. When a change occurs in any of the monitored items, a notification will be published by the server (OPC Foundation, 2017).

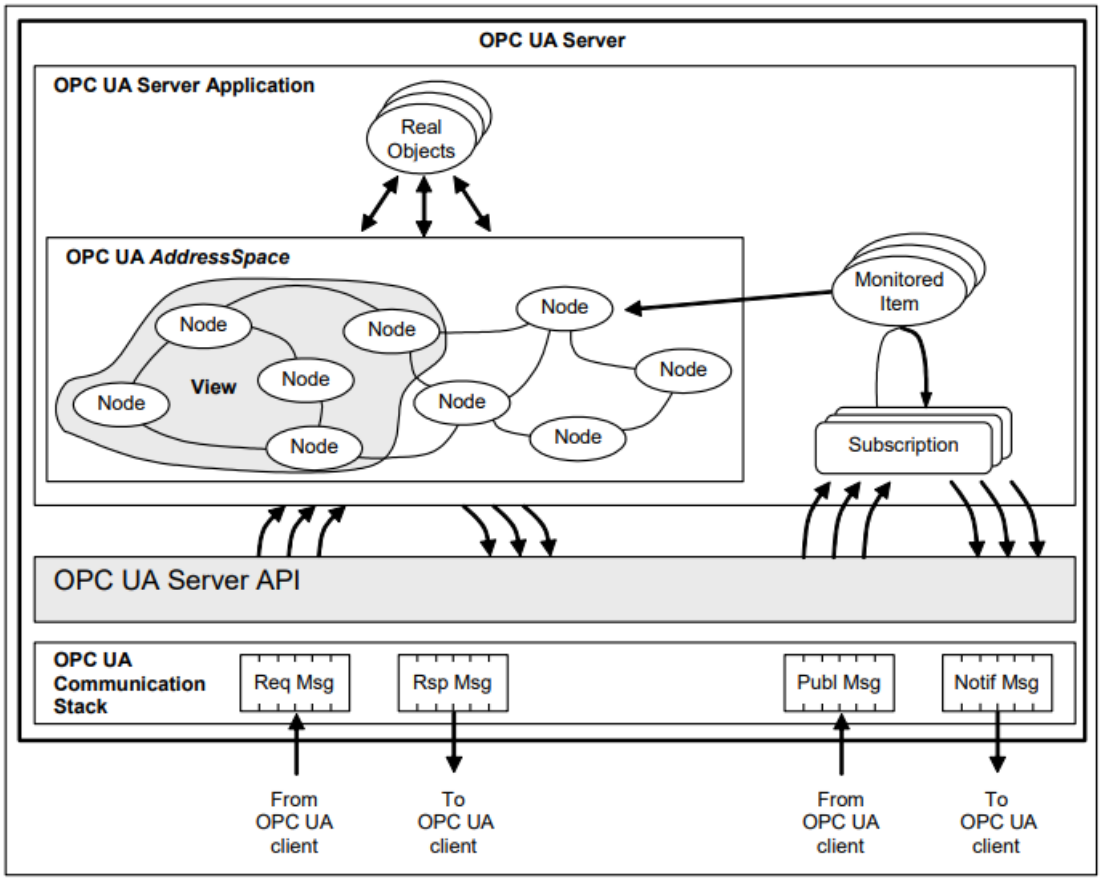

Figure 3 A brief overview of how an OPC-UA server operates (OPC foundation, 2017).

2.1.1 Server discovery

For modular usage, it is preferable to be able to dynamically connect modules that will be automatically discoverable in the network. For this reason, the OPC foundation has specified how applications can discover servers and connect to them automatically. There are three discovery server types: Local Discovery Server (LDS), Local Discovery Server with Multicast Extension (LDS-ME), and Global Discovery Server (GDS). The LDS can discover registered applications that exist on the same host. The LDS-ME can discover applications announced on a Multicast Subnet (e.g. a Local Area Network (LAN)), where all hosts receive the same packets. With GDS, clients can find servers in the same administrative domain.

If an OPC-UA server needs to be discoverable, it must be registered as a server to a dedicated discovery server. The discovery server maintains information of all registered servers. When a client wants to discover servers in the network, it sends out a request to the discovery server, who in return responds with information of the registered servers and how to connect to them. Once a client is connected to a server, the server information is stored by the client. Thus, the next time that the discovery procedure is executed is when the client is unable the reach a known server.

In some cases, a host might only have one OPC-UA server (e.g. embedded systems). Instead of implementing a dedicated LDS, the host can act as an LDS-ME by announcing it to the Multicast Subnet. For this method, a Multicast Domain Name System (mDNS) must be utilized by the network. This simply means that the servers announce themselves as servers in the network and respond to incoming mDNS messages from other hosts in the network (OPC Foundation, 2018).

2.1.2 Time-Sensitive Networking (TSN)

In industrial automation, usage of Ethernet-based bandwidth is increasing every year. Due to more complex processes with real-time requirements in Industry 4.0, the traditional Ethernet networking protocol has its limitations. In the Ethernet's infancy,

it was designed to manage all traffic equally. Thus, time critical and none-time critical traffic cannot be distinguished from each other.

TSN is a set of standards that as of writing this thesis is being determined, with a preliminary goal to be finalized in 2020. This is to be implemented as an extension to its predecessor IEEE 802.1, which is a standard for Ethernet Local Area Network (LAN). TSN utilize Gigabit Ethernet to achieve time critical messaging. In addition to the time critical aspect, TSN also ensures enhanced reliability, fault tolerance, and security. In the TSN, time is synchronized to all devices with a maximum deviation of 100µs (Bello & Steiner, 2019). When several frames are sent over TSN, the frame with the highest priority is processed first to ensure that the time-critical frame arrive on time. If a low priority frame is in the process of being sent, a higher priority frame has the capability of interrupting the ongoing lower priority frame. TSN can function with non-TSN Ethernet, but to interpret TSN frames correctly, the switch (also called bridge) in a TSN network must be compatible with TSN standards. With the gained reliability and low time latency, applications that previously were not considered reliable or low latency enough over network protocols, such as safety for motion control, can now be used (Bruckner et al., 2019).

Vendors are in the process of developing hardware that support TSN capabilities. In a theoretical comparison of OPC-UA TSN with current Ethernet-based industrial communication systems (Powerlink, Profinet, and EtherCat), the authors found that OPC-UA TSN is up to 18 times faster (Bruckner et al., 2018). An illustration of this can be seen in Figure 4. An experiment with a line topology containing 50 Input/Output-devices from manufacturer B&R, resulted in a 50ns standard deviation of time synchronization and a cyclic-time of 50µs. By the authors' estimation, up to 200 devices in a single line topology would still be able to achieve a cyclic time of 50 µs (Bruckner et al., 2018).

For the past half a century, it has been envisioned of a single uniform and standardized industrial communication protocol. For a first time, top vendors in the industrial automation market all agree that OPC-UA TSN is a superior protocol that should be

de facto standard communication protocol of the future. (Bruckner et al., 2018).

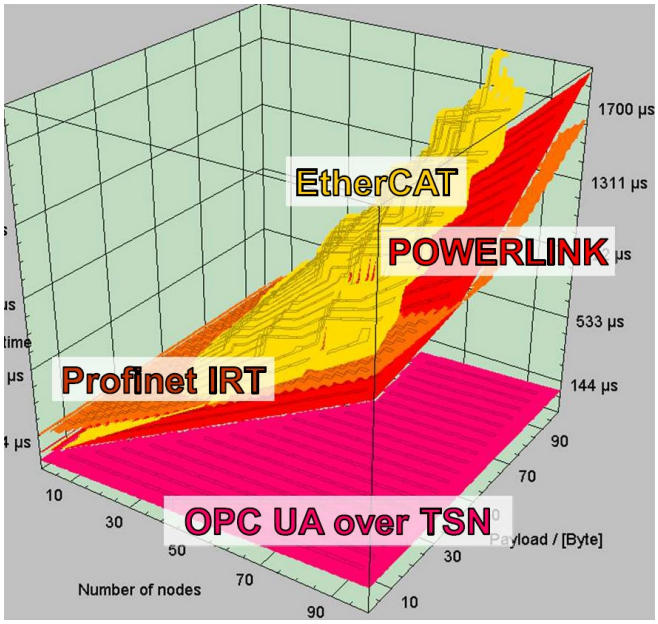

Figure 4 OPC-UA TSN with Gigabit physical layer outperforms current solutions (based on 100Mbit) up to 18 times (Bruckner et al., 2018).

2.2 Graphical user interface

In an industrial environment several types of GUI's (Graphical User Interfaces) are used. The implemented GUI type is chosen based on the environment and the task needed to be monitored/controlled.

In a factory's production facilities, there might be machines that workers need to interact with in order to sufficiently supervise the production process. Traditionally, this supervision would be a device with a GUI connected to a PLC, where both devices are from the same vendor, and thus uses vendor specific software and communication. Within a factory, a device with the GUI is commonly an operator terminal, which is further described in Chapter 2.2.2. The communication between the operator terminal and the PLC is the communication between the 2 first levels of the automation pyramid mentioned in Chapter 2.1. The GUI is preferably simple to use and should not contain more functions than is necessarily needed. This is quite straight forward to implement and is sufficiently robust. But what would happen in a scenario where ten years after

a powerplant has been commissioned, one of these GUI devices suddenly cease to function properly, or in a worst-case scenario cease to function completely. If both devices have vendor specific software to communicate and function with one another, the only alternative when replacing a faulty device is to replace it with a spare part from the same vendor. The catch here is that vendors will not ensure that their device that was once purchased, will still be available for purchase ten years down the line. If the vendor does offer compatible spare parts, the vendor might have a hefty price due to no longer being a part of the vendors official product portfolio.

When there are several processes within an industrial facility, a centralized monitoring system is commonly implemented, where all individual processes can be monitored and managed in case of unexpected anomalies. This centralized monitoring system is commonly known as Supervisory Control and Data Acquisition (SCADA). Wärtsilä has developed a custom SCADA-system called Wärtsilä Operator Interface System (WOIS). In a control room, there are usually several PC's connected to the SCADA-system. As mentioned in Chapter 2.1, OPC is a common communication protocol used to communicate between PLCs and the SCADA system. The backbone in a SCADA system is based on a server model. User interfaces as well as historical data are stored on servers, so data can be accessed over TCP/IP. A common communication standard in Europe for this is IEC 60870-5 (Jayasamraj, 2011). Depending on the number of devices needed to be monitored, the requirement of the servers increases accordingly. In a larger powerplant, there can be one to several server-cabinets containing several racks. On the contrary, for smaller powerplants a smaller server setup might be enough. For a smaller setup, the capital cost of servers and licences might be a significant percentage of the total costs of the powerplant, thus not the most cost-effective alternative.

With the emergence of new capabilities of industrial hardware, it has become more appealing to diverge from a centralized SCADA system for smaller powerplants and instead utilize a decentralized model. In a decentralized model, the GUI and data storage are distributed throughout the powerplant yet monitoring of the powerplant can still be performed from a traditional control room.

2.2.1 Webserver on process controllers

Implementing a webserver on a PLC is nothing new. In fact, it has already been possible for at least a decade. These webservers have mostly been utilized for remotely accessing datafiles stored on the PLCs memory, using File Transfer Protocol (FTP).

In an experiment by Kopček (2016), a Siemens S7 PLC was used to demonstrate the possibility of storing GUI source files (HTML, XML, CSS, and JavaScript) on a webserver in a PLC. Clients were subsequently able to access the GUI with a standard web browser over HyperText Transfer Protocol (HTTP), or alternatively the secure end-to-end encryption (HTTPS). These source files are a set of machine-readable datafiles, meaning that the instructions in the code is interpret by a machine without the need to compile the code first. This is comparable to browsing the Internet, where the source files for a webpage are stored on a server somewhere in the world. A client can request the source files from a server and subsequently receive the source files, to display the webpage. To access process values (i.e. machine data on the PLC runtime), a communication must be established between the runtime data and the webserver. Siemens has a set of HTML functions that can access values from the PLC's runtime system, which was used in the experiment by Kopček (2016). An alternative method that vendor B&R uses, is that the machine data and the webserver communicate over OPC-UA, so clients subsequently can access the webserver over HTTP/HTTPS. An illustration can be seen in Figure 5, where two different communication methods needed. A wide variety of vendors currently have webserver capabilities on at least some of vendors process controllers.

Figure 5 The topology of the communication to and from a webserver.

Creating HTML5 source files, requires that the developer knows how to write source code for HTML5 web pages, which can be a time-consuming process. Although there are programs such as Adobe Dreamweaver for creating web pages, the developer must code manually. To simplify the creation of source files, vendor B&R has created an intuitive semi-graphical development tool called mappView, which is further described in Chapter 3.1.2.

Throughout a powerplant, there are usually several process controllers (i.e. PLCs) that each control local tasks. These PLCs forward information to a SCADA system. To make the GUI decentralized in a powerplant, one solution would be to store the GUI source files locally on the PLCs throughout the powerplant. If all devices are connected on the same network, the GUIs can be accessed from anywhere within the powerplant.

If historical data storage is desired, an industrial PC can be used instead of a PLC. An industrial PC can utilize a hypervisor (described in Chapter 2.3), to manage two operating systems (OS). The first OS has a real-time system for process control, where a webserver for GUI source files can be implemented as well. The second OS is a general-purpose operating system (GPOS), where a database management system (DBMS) can be implemented, as well as any other type of application for a GPOS.

2.2.2 User interface hardware and software

An operator terminal is either a touchscreen or a display with cohering buttons in a robust enclosure, intended to be used in an industrial setting. In the past, the norm was that each vendor had their own software for configuring the terminals with graphics and mapping the process values. The terminal was then limited to only communicate with process controllers by the same vendor. When these devices are implemented in an automation system, eventual component replacement in the future is thus limited to use the same vendor’s component.

Recently, industrial automation vendors have started to develop terminals supporting HTML5 web browsing capabilities. Due to the availability, it becomes more appealing implementing webservers on process controllers, where the GUI source files are stored, as mentioned in Chapter 2.2.1. A huge advantage of this method is the confidence of the availability of hardware that support HTML5 in the future. Regardless of which vendor is offering the best price, the HTML5-based terminal will function properly.

With the introduced possibility of using Cascading Style Sheets (CSS), the user interfaces can be created with more advanced graphics. As a result of more advanced graphics, the requirement of the terminal hardware increases accordingly.

An alternative to using an operator terminal is to use a panel PC (PPC). A panel PC is similar to the industrial PC mentioned in Chapter 2.2.1, but the difference is that the panel PC consists of an industrial PC and a touch panel encapsulated in one unit. The panel PC usually has better hardware capabilities than an operator terminal, making the panel PC more likely to manage user interfaces with more advanced graphics without any major response delays.

As mentioned in Chapter 2.2.1, an industrial PC can be utilized instead of a PLC, so a database management system can be implemented. If a local database is required, and a panel PC is used instead of an operator terminal, the industrial PC is not necessarily needed, since the database can be implemented on the panel PC instead of the industrial PC.

The performance of the devices can vary, depending on which web browser application is used. In an operator terminal, the web browser application cannot be chosen by the end user, rather the vendor of the operator terminal has chosen the web browser application when the operating system for the operator terminal was developed. On the panel PC, the end user can choose the web browser application of their choice.

In experiments by Radhakrishnan (2015), the most common web browser applications were benchmarked. The three tested web browsers were: Microsoft Internet Explorer (IE), Mozilla Firefox, and Google Chrome. A total of 16 different benchmarks were performed, grouped into four categories: Core Language Features, Data Manipulations, Graphics Emulation, and Compiler Efficiency. When limiting the number of CPU cores used, the author found that the browsers did not make use of any additional cores beyond two. When varying allocated memory (i.e. Random-access memory (RAM)) to the browsers, only Firefox gained performance with additional memory. However, the performance of Firefox at max allocated RAM was equivalent to the performance of Google Chrome. The benchmark scores of limited CPU cores and available RAM, can be seen in Figure 6.

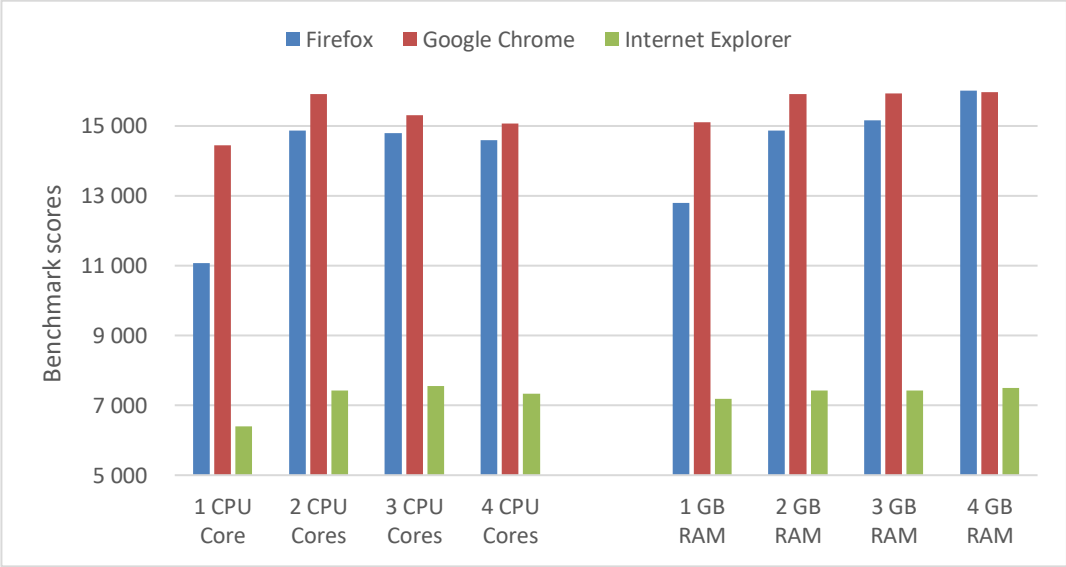

Figure 6 Benchmark scores of web browsers with limited hardware resources (Radhakrishnan, 2015).

With the number of cores set to two and the RAM limited to 2 GB, 16 separate benchmarks were performed. The benchmark scores of these tests can be seen in Figure 7, where Firefox performed better in data manipulation and graphic emulations, while Google Chrome performed better with compiler efficiency and core language features. The Internet Explorer is not an efficient web browser, when compared to the other two tested browsers.

As of this writing, the experiments by Radhakrishnan (2015) were conducted a few years ago, so if the experiments were to be replicated, one would most likely get a different outcome. From these tests however, one can compare how the different browsers perform in relation to one another in different categories.

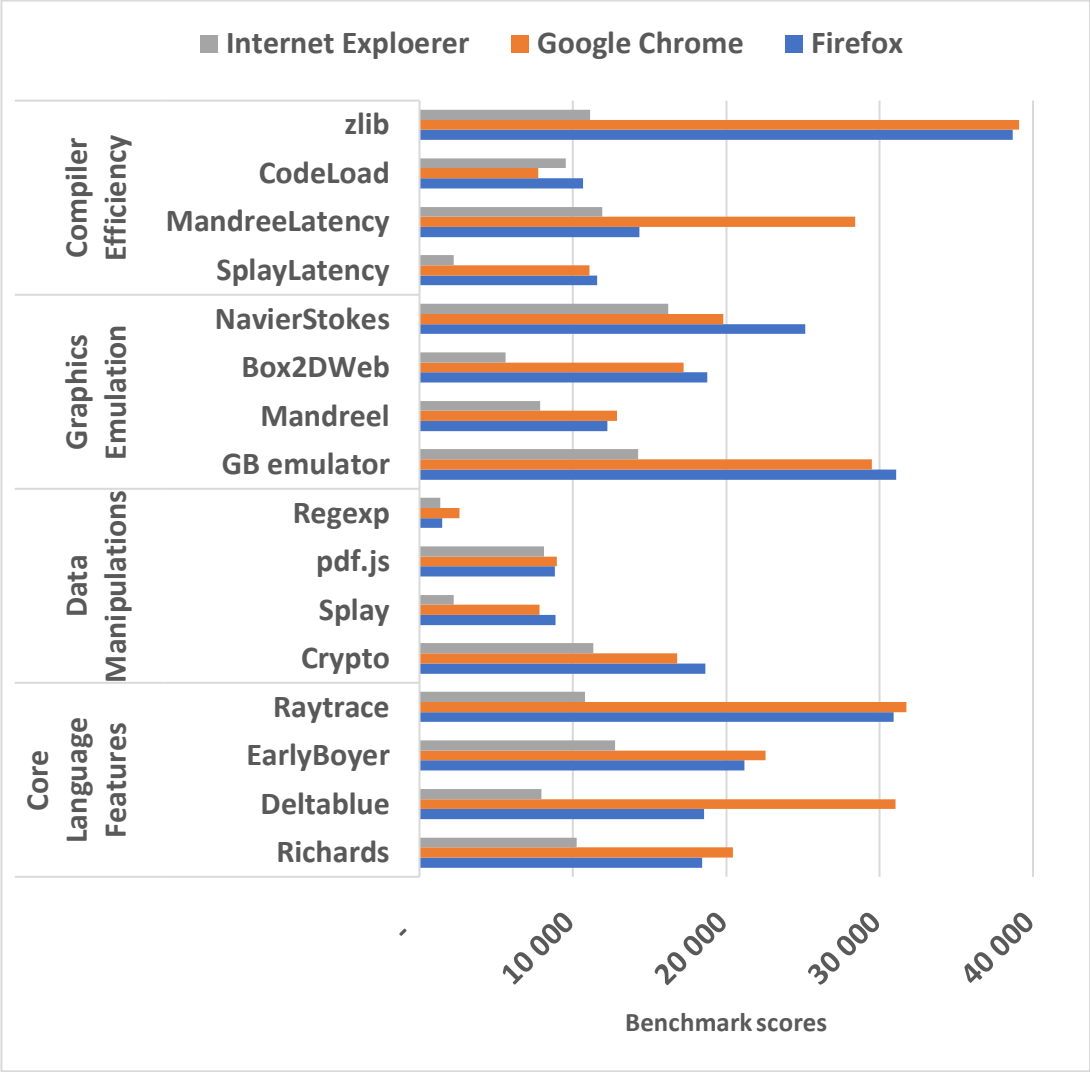

Figure 7 Comparison of web browsers when benchmarked in different categories (Radhakrishnan, 2015).

2.2.3 External communication and security

Connecting all devices to the same TCP/IP network, enables the possibility to connect the devices to the Internet. This would enable the customer to remotely monitor the powerplant from anywhere in the world. This is of course a neat feature, but the

security aspect should also be taken into consideration.

It has become important with cybersecurity in today's industry, since an increasing number of devices are connected to the Internet to be utilized for IIoT. An industry may suffer from devastating damages if any device in the network is subjected to malicious malware. There have been many of these attacks in the past years. The best solution to avoid external attacks is simply to be disconnected from the Internet. However, devices can still be affected by malicious malware, regardless if the devices are connected to the Internet or not. An example of this occurred in 2010, when a nuclear powerplant in Iran was targeted with a virus named Stuxnet. Stuxnet was automatically copied to USB-drives, when a new USB-drive was detected. When a USB-drive containing Stuxnet was inserted into a USB port of a computer, Stuxnet proceeded to scan the network for Siemens S7 PLCs that controlled frequency converter drives managing motors between 807 Hz and 1210 Hz. This is a uniquely high speed for motors, thus only effecting a small set of devices. Once Stuxnet found the right target criteria, the frequency for the motors were altered over a period of several months (Falliere et al., 2011). Due to the frequency alternations, 984 uranium enriching centrifuges were destroyed (Holloway, 2015). Thus, ensuring that unauthorized people are unable to access the mainframe is just as important as cybersecurity.

If the decision is made to add Internet connectivity to the powerplant's network, appropriate security measures should be established. The OPC-UA protocol (see Chapter 2.1) was developed so messages can be encrypted, thus an appropriate protocol for transferring process information between devices in a powerplant. Furthermore, the TSN mentioned in Chapter 2.1.2 amplifies the security of the OPC-UA protocol. Clients that accesses the GUI files stored on the PLC, can communicate with HTTPS for an ensured end-to-end encryption. If the client device is a panel PC, with a DBMS is implemented on the panel PC's GPOS, while the GUI is stored on the panel PC's real-time OS, the two operating systems can communicate with one another through a virtual port, thus ensuring that the communication is secure from external attacks. Streaming replication of a databases (see Chapter 2.4.3), supports SSL-encryption between a primary and replica server.

2.3 Hypervisor

A hypervisor is a firmware, on which virtual machines operates. Virtual machines on the hypervisor are known as guests, each with their own instance of an operating system. There are two types of hypervisors: hosted and bare-metal. The hosted hypervisor operates on an GPOS, such as Windows, while the bare-metal hypervisor operates directly on the host machine's hardware. In Figure 8 the two hypervisor types are illustrated.

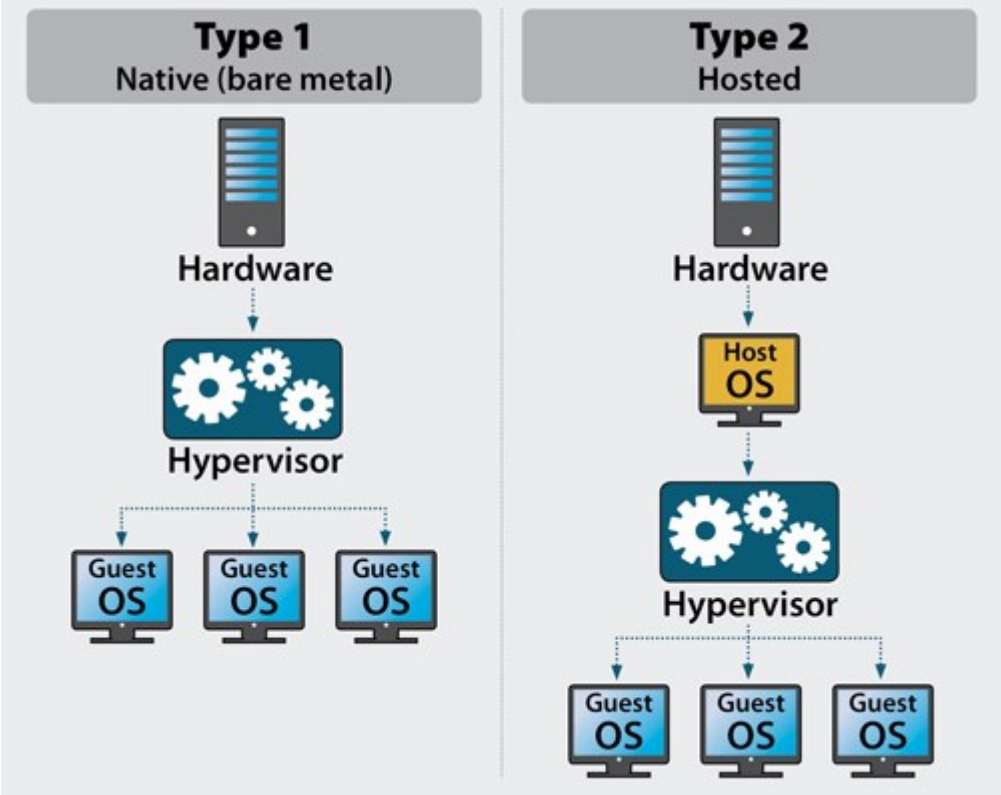

Figure 8 The difference between a bare-metal and a hosted hypervisor (Taylor, 2018).

The disadvantage of the hosted hypervisor is that the guests depend on the host operating system. If the host operating system crashes, all the guest's operating systems crashes as well. Since the bare-metal hypervisor operates directly on the host's hardware, all the guest operating systems can function independently without interfering with one another. The bare-metal hypervisor physically allocates hardware recourses to each guest. For example, each of the host processor's cores are allocated to separate guests.

In most cases, process controllers (i.e. PLCs) with a real-time operating system are used in the automation industry. In addition to the real-time system, it has become more common to also implement a general-purpose operating system, to manage tasks such as: databases, analytics, visual recognition, and computationally intensive user interfaces. With a bare-metal hypervisor, two operating systems can be implemented in one device. Since the lack of interference with one another, a real-time system can be assured for the controller. Although the guests operate independently, a virtual port can be established to communicate with one another, transferring data between applications on the two operating systems (B&R, 2018).

2.4 Database management system (DBMS)

These days the term Big Data (i.e. collecting valuable data for analysis), has become quite common. This is nothing new in Wärtsilä's powerplants. Wärtsilä's powerplants offer services to store and manage process data for up to ten years in Wärtsilä Operator Interface System (WOIS). With a sampling rate of one second and roughly 400 measurement points, each with several properties, this accumulates to roughly 300 billion datapoints (300×10^9) over the course of ten years. To manage this large amount of data, a database management system is preferably chosen.

2.4.1 Relational and non-relational database

There are significant differences between the two databases depending on where and how they are used. The relational database is the traditional database that was invented by E.F Codd in 1970, in which the data is organized in a table format with columns and rows. To access the relational database, the Structured Query Language (SQL) is used. Originally this was based on relational calculus and algebra, thus given its name. The non-relational database, also known as NoSQL, was introduced in the late 90's when more complex data needed to be stored, such as key values, objects, and documents. The NoSQL, in comparison to the relational database, is not based on a table structure, and thus neither use the aforementioned SQL. However, there are exceptions of instances where a non-relational database uses SQL. The relational

database has a few shortcomings, such as it does not scale easily, and when there is a huge amount of data, it must be partitioned on several servers (Jatana et al., 2012).

When it comes to the performance of these, it can be quite case specific. In a comparison, Li & Manoharan (2013) found that various NoSQL databases do not always necessarily outperform the relational database. This is confirmed by van der Veen et al. (2012) who state that none of the tested databases outperform each other in both read- and write operations. Focusing only on intensive write operations, Fatima & Wasnik (2016) found that the NoSQL has a clear advantage.

2.4.2 Time-series database

A time-series database can contain a variety of values, but the primary content is the timestamp of each insert. The key importance of a time-series database is that it is optimized to see how values change over time. This is typical for an industry collecting sensor data with a defined sample rate, after which the data is periodically inserted into a database. However, this database is hardly ever updated, instead inserts occur periodically and deletions are performed in bulk. With a time-series database, one can perform several types of analytics and query specific datasets with defined SQL functions. This makes it possible to collect data with a sample rate of a few milliseconds, yet data can still be quickly analyzed over several months (Microsoft, 2018).

There are several types of time-series database management systems. Two popular ones are InfluxDB and TimescaleDB. Although both are open source time-series databases, they are vastly different.

InfluxDB is a NoSQL database that was purposefully built from scratch as a time-series database (Influxdata, 2017). On the contrary, TimescaleDB is an extension to PostgreSQL, which is a relational database. TimescaleDB was introduced in 2017, while influxDB was created in 2013. However, PostgreSQL was introduced in 1996 so TimescaleDB has the advantage of standing on the shoulders of giants, so to speak.

PostgreSQL has over two decades of experience with reliability and security. TimescaleDB supports traditional SQL and has many additional functions. Due to TimescaleDB's support for traditional SQL, a third-party software can be easily implemented. However, InfluxDB has its own modified SQL, making it more limited to third-party software.

In a case study comparing InfluxDB and TimescaleDB, Tallkvist & Warrén (2019) found that TimescaleDB performed significantly better when used to store sensor data with a high sampling rate. In both cases of reading from and writing to the database, TimescaleDB outperformed InfluxDB by up to 6000%.

2.4.3 Database reliability

When evaluating the reliability of databases, PostgreSQL can be presented as an example. PostgreSQL has a tool called streaming replication. As the name suggests, one or more replicas of a primary server can be established. When a database is to be modified (i.e. insert, update or delete), the instructions are individually written to a Write Ahead Log (WAL). The WAL is streamed to all replicas of the database. Once the WAL has been written and streamed to each replica, the changes are applied for each of the servers. Each of the replications can exist on different physical machines, to ensure reliability. In case one of the machines fail, data is safely stored on the remaining machines. An illustration of this can be seen in Figure 9. While applying the changes in the WAL to the servers, checkpoints are periodically set. If any of the servers crash while applying the changes, then upon rebooting, replication can continue from the last set checkpoint.

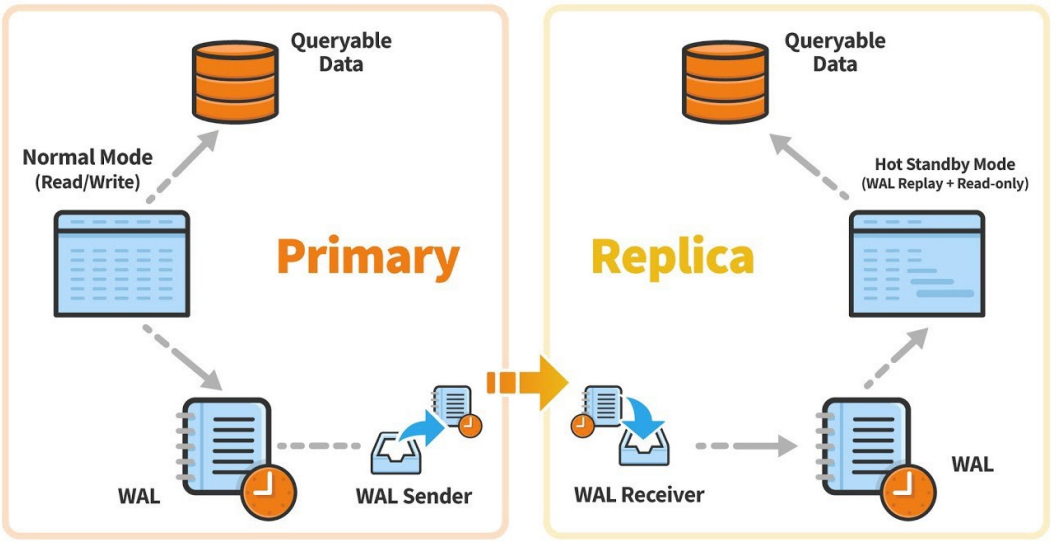

Figure 9 The WAL is sent from the primary server to all replicas where the instructions in the WAL are applied to the servers (TimescaleDB, 2018).

The advantage of writing instructions to the WAL instead of directly writing it to the main data files, is that the WAL is written sequentially to the disk. Sequential write operations mean that large contiguous blocks of data are written to adjacent locations on the disk in the hard drive, which is significantly faster than random writes that may be spread across multiple sections of the disk. Even if a solid-state drive is used, sequential writing is a superior method.

There are four methods in which a database replication can be configured. Listed from highest risk of data loss to least: (i) no replication, (ii) asynchronous replication, (iii) synchronous write replication, and (iv) synchronous apply replication. In method (i), where no replication is used, data is lost if the primary database is corrupted. In method (ii), asynchronous replication sends the WAL to all replicas, but the replicas do not confirm if they have received the WAL or not. Instead, the write command of asynchronous replication is considered successful once data has been written to the primary server's WAL. In method (iii), the synchronous write replication guarantees that all replicas have received the WAL, before reporting a success back to the client. Lastly, in method (iv), not only does synchronous apply replication ensure that the WAL have been received by the replicas, but it also ensures that the changes in the WAL have been applied to all servers.

When no replication is used, the highest data insert rate can be achieved, but is more vulnerable to data loss. On the contrary, when synchronous apply replication is used, the data insert rate is reduced roughly by half, but a significantly smaller risk of data loss can be assured (TimescaleDB, 2018). The difference of the replication methods insert rates, can be seen in Figure 10.

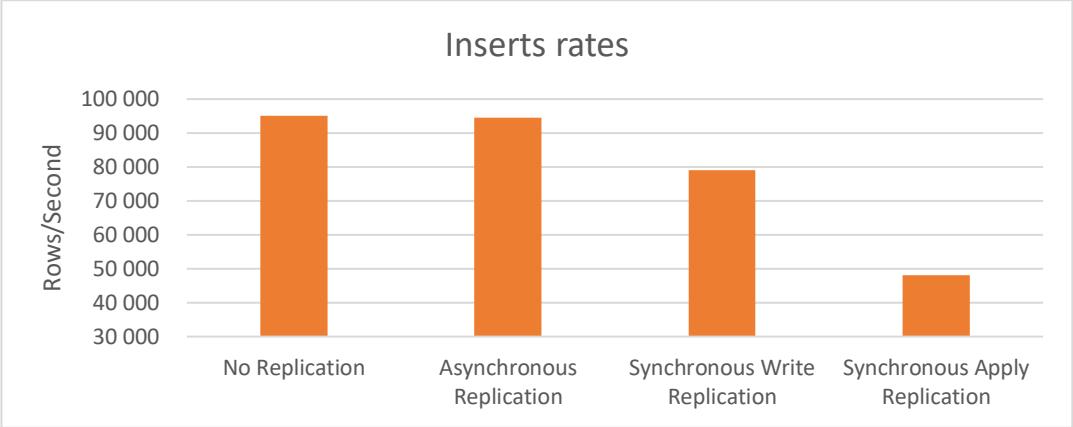

Figure 10 The rate at which inserts are executed depending on the replication methods in use (TimescaleDB, 2018).

Replication can be further classified into three different configuration modes: physical replication, logical replication, and clustered physical replication. Physical replication replicates the entire primary database and mirrors that to the replica main database. If several replications are desired on a single server, physical replication cannot be used. This topology is illustrated in Figure 11. Fortunately, many replications on a single server can be achieved by both logical replication and clustered physical replication

Logical replication uses a publish/subscribe method. Unlike physical replication, logical replication does not replicate the entire database, rather only one to several tables of a database. When a primary database applies changes to a table, these instructions are published. Subsequently, clients can subscribe for these changes and applies the same changes on the replication table. When a new subscription of a table is created by a client, a snapshot of the table on the primary database is replicated to the client. Logical replication requires that the structure of the client's table is identical to the table structure in the primary database. When changes are applied to a table configured for logical replication, the aforementioned WAL is utilized to publish the

instructions to the subscribers (severalnines, 2018).

The third replication mode (i.e. clustered physical replication) is illustrated in Figure 11. Clustered physical replication is similar to regular physical replication, in the sense that the entire primary database is replicated. On the replica machine however, the main database manages one to several clustered databases. The main database on the replica machine, as well as all the clustered databases, communicate on unique networking ports, enabling each of the databases to communicate independently from one another. Each of the clusters are registered as independent services to the replica machines operating system. Similar to physical replication and logical replication, a WAL with instructions is sent from the primary database, when the clustered physical replication is used. The clustered replicas receive the WAL from their cohering primary database and applies the WAL instructions to the replica database (PostgreSQL, 2019).

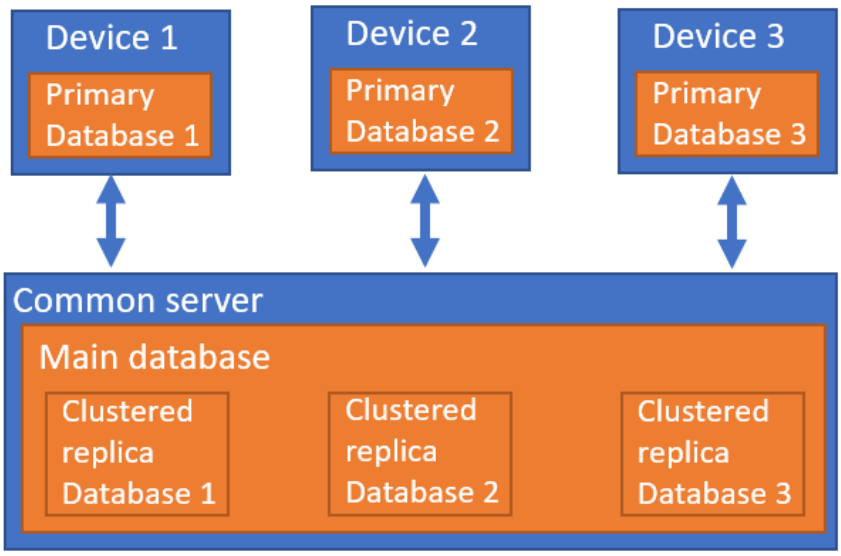

Figure 11 A server with clustered database replicas from several individual primary databases.

In a study of fault tolerances for off-the-shelf SQL databases, bug reports submitted by database users, where tested on various DBMS. Out of 273 bugs, very few of them affected two different databases (Gashi et al., 2007). Thus, additional reliably can be achieved by storing data on more than one type of database, for example by storing identical data on both InfluxDB and TimescaleDB.

When taking into consideration the physical condition of the storage medium, failure is inevitable. The disk in hard drives, as well as solid state drives, has a finite lifetime. Therefore, complete redundancy can only be achieved by backing up data periodically or utilizing a Redundant Array of Independent Disks (RAID). There are 7 levels of RAID (from 0 to 6), where RAID 1 mirrors all data between two different drives. In case one of the drives fail, the faulty drive can be swapped out for a new one, after which all data is copied from the functioning drive to the newly inserted drive (Chen et al., 1994). This is also known as hot swapping, where components can be exchanged without shutting down the machine.

2.5 The engine in a compact powerplant

This chapter focuses on Wärtsilä's engines that are used in modular powerplant solutions, as oppose to larger powerplants where a set of larger engines are installed next to each other in engine halls. The control system for the smaller modular powerplants, is the basis of this thesis. The needed graphical user interfaces to control the engines, is described in Chapter 2.2.

2.5.1 W20 Containerized powerplant

The basis for the modular powerplant is based on a 40-foot high-cube container. The containerized powerplant is pre-fabricated in these containers and can be shipped to the customer on a regular container ship. At the installation site, a concrete foundation needs to be constructed before assembly of containerized powerplant can begin. The advantage of this modularization is that after its arrival to the customer, the powerplant can be commissioned in a few weeks, making the commissioning duration much faster than a traditional engine powerplant. Due to the only fixed structure being the concrete foundation, if needed, the powerplant can be relocated after it has been commissioned.

The engine used is a W9L20, which can generate 1.6-1.9 MW, depending on the engine rpm and the type of fuel used. The engine and generator are fitted in one container and cohering auxiliary equipment is placed in an adjacent container. This is

referred to as a generating set, which is shortened to genset. The number of gensets at one site can be scaled up according to the customers energy need. In Figure 12, three gensets are illustrated. When several gensets are deployed, a common module is optionally implemented, from where the gensets are monitored. The W20 containerized powerplant is no longer part of Wärtsilä's official product portfolio. However, if a customer expresses specific interest in the containerized powerplant, it may be sold.

When these containerized powerplants have been sold in the past, the supervisory system WOIS has been implemented on a computer with Microsoft's Windows 7. Recently though, Microsoft has terminated the support for Windows 7. Due to this, if containerized powerplants are sold in the future, a server-based WOIS will be implemented instead of the Windows 7-based WOIS. However, the findings from this thesis presents an additional method to be used instead of the server WOIS.

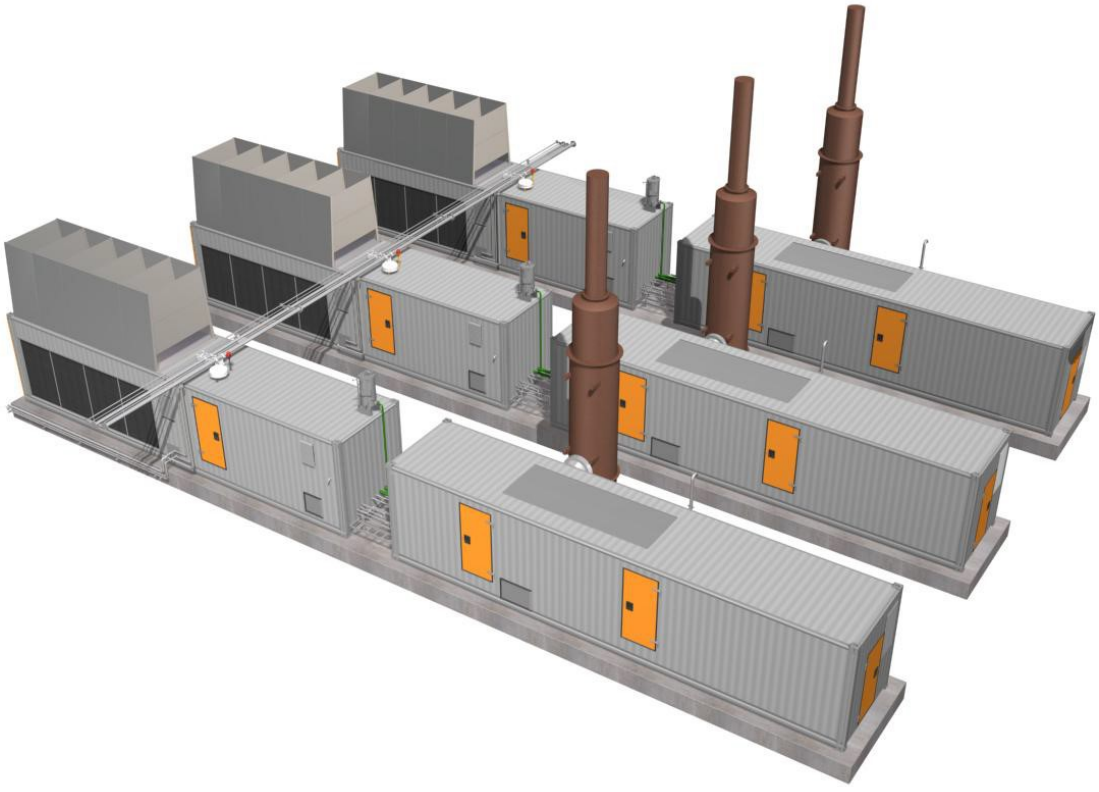

Figure 12 Three generating sets, where the three modules on the right hand side contains the motor and generator and the remaining three containers on the left hand side contain auxilary equipment (Wärtsilä, 2010).

2.5.2 Wärtsilä Modular Block

Similar to the W20 containerized powerplant mentioned in Chapter 2.5.1, the Wärtsilä Modular Block is based on prefabricated containers. Both powerplant types are similar in the sense that a concrete foundation is needed at the installation site, before assembly of the powerplant can begin. Due to the limitation of a containers internal volume, only a smaller sized engine will fit inside. The efficiency of the smaller sized W20-engine is not as good as the larger sized high-efficiency medium-speed engine. Thus, Wärtsilä Modular Block uses the medium-speed engine W34SG that has an output power of 5.5-9.8 MW, depending on the configuration. Currently, the W32HFO-engine is in development to be implemented as an option instead of the W34SG-engine. The W34SG is a gas fuel engine, while the W32HFO runs on heavy fuel oil. On the contrary to the W20-engine, the W34SG engine does not fit inside a high-cube container, instead the containers only comprise the necessary auxiliary equipment. At the installation site, pre-fabricated containers are bolted together and forms the outer walls around the engine. Only four walls are shipped to the installation site, so if several engines are sold, the engines are placed next to each other, so only the outer most walls are needed. An illustration of a setup with one engine can be seen in Figure 13, where one of the side walls have been removed to reveal the engine. This engine facility is to be unmanned during normal operation, thus a separate facility is additionally needed, from where the powerplant can be remotely supervised (Wärtsilä, 2019).

The Wärtsilä Modular Block has recently been added to Wärtsilä's powerplant portfolio. The supervisory system chosen for Wärtsilä Modular Block is the server-based WOIS. The results from this thesis to evaluate a light WOIS, could be an alternative system for Wärtsilä Modular block in the future.

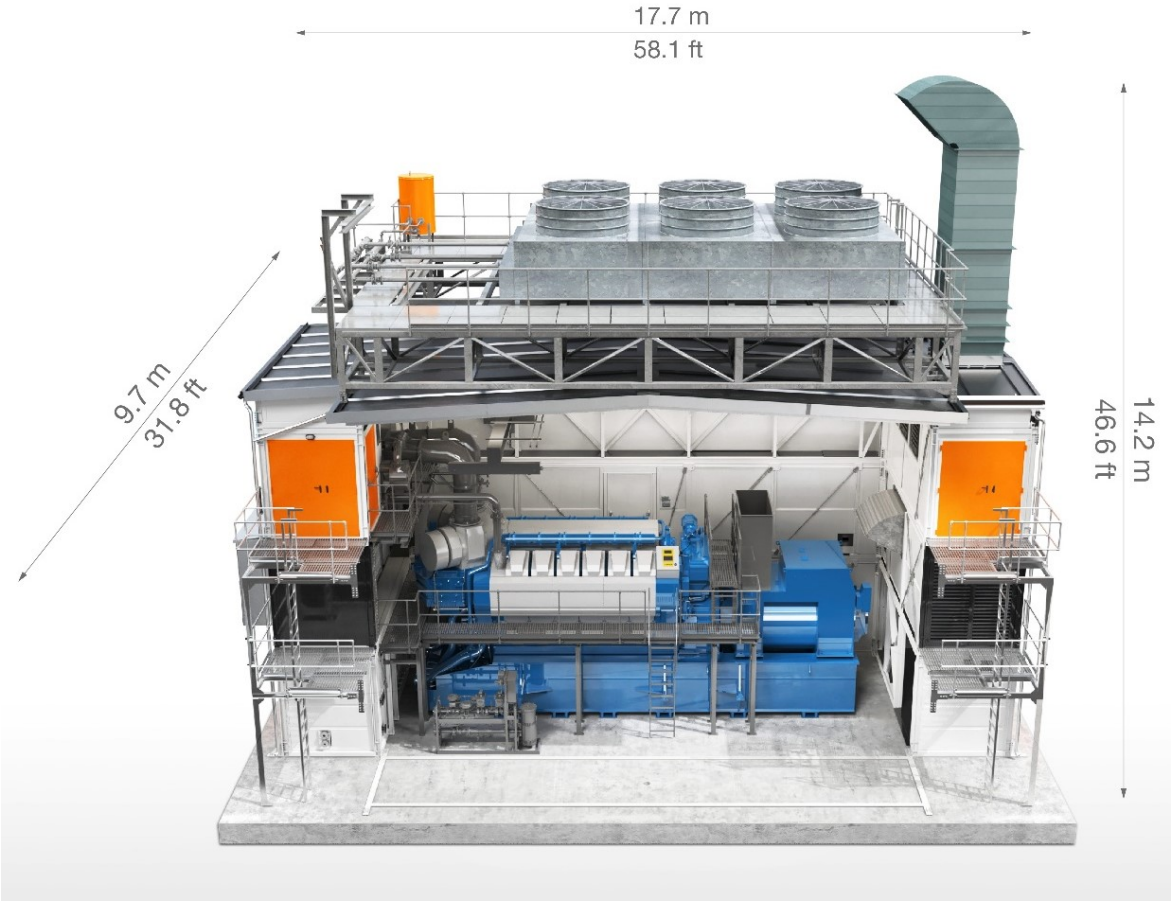

Figure 13 Wärtsilä Modular Block, where one wall is removed to reveal the W34SG engine.

2.5.3 Unified Control (UNIC)

UNIC is an embedded management system for the engine. The system manages start/stop, engine safety, fuel control, speed/load control, cooling, and combustion. UNIC is a modular system that have been developed to withstand vibrations, so the modules can be mounted directly to the engine for a compact design. A variety of sensors and actuators are connected to UNIC for monitoring and controlling the engine.

To monitor and control the engine from a control room (i.e. with WOIS), the UNIC system communicate with an external process controller, whos main task is to manage the auxiliary equipment. This communication is either with Modbus TCP or OPC classic. The process controller subsequently relays the information to WOIS.

The aforementioned communication from the engine to the process controller has already been translated to modern OPC-UA but is still under evaluation. Once the OPC-UA communication from the engine is released to the engine production, the external process controller will no longer be required to translate the engine's process data. Any node in the network with OPC-UA capability, will be able to read the engine data. Even if the communication is switched to OPC-UA, the PLC in the auxiliary module is needed for the local auxiliary equipment.

3 PROOF OF CONCEPT OF DECENTRALIZED AUTOMATION

In recent years, automation vendors have released new revisions of their development software, with a semi-graphical editor to create HTML5-based user interfaces. With this editor and pre-developed libraries of widgets, developers with no prior knowledge of hypertext-based coding can now relatively quickly create HTML5-based graphical user interfaces. Although many vendors are now emerging with HTML5 editors, B&R was at the forefront quite early. B&R announced the HTML5 editor in a press release November 2015. In this thesis, B&R's development tool Automation Studio with the application mappView, is evaluated for creating the HTML5-based user interfaces. The objective was to mimic the currently utilized WOIS and create a lighter HTML5 version.

To enable the user interface to store historical data for longer periods of time, a third-party database management system (DBMS) additionally needs to be implemented. In this thesis, the DBMS PostgreSQL with a time-series extension TimescaleDB is evaluated.

3.1 Automation studio

The Automation Studio program is used for configuring and programming B&R's products. The standard IEC 61131-3 programming languages are supported, as well as ANSI C/C++. Once a project is compiled in Automation Studio, it can be transferred over TCP/IP to B&R's hardware with Automation Runtime, which is a real-time operating system developed by B&R.

Automation Studio has a set of functions, mutually named mapp technology (modular application technology). Mapp is part of the application layer and supports developers of machine and system applications by providing a way to efficiently implement recurring functions. The emphasis here is on flexibility, scalability, quality and an easy-to-understand interface.

3.1.1 mappAlarmX & mappAudit

To monitor process data and indicate to the operator when there are anomalies, mappAlarmX was developed. This application can monitor and maintain current alarms, as well as storing historical alarms that already have been acknowledged.

The mappAudit application is similar to mappAlarmX. However, mappAudit is more applicable for events, such as user interactions. With a list of occurred events, it can be easily tracked for example, when a motor was started, or when a setpoint of a process value has been changed by an operator.

To enable mappAlarmX and mappAudit, they are added to the Automation Studio project respectively and configured with monitored process values and their operation limits, along with metadata such as name, message, code, severity, and behavior.

Current alarms, historical alarms, and audit events can all be queried based on specifics in metadata or the timestamp of the instance. Alarms and/or events from specific tasks can thus be filtered and displayed for task specific sections (i.e. in a user interface). Historical alarms and Audit Events can be exported as comma-separated values (CSV), to either a USB-storage device or to the PLC's user storage area. If exported to the latter, these files can be accessed with FTP.

Historical alarms, current alarms, and Audit events can be easily implemented in a HTML5 user interface. In mappView (see Chapter 3.1.2), there are pre-developed widgets for alarm applications. Once mappAlarmX and mappAudit is configured, mappView can access lists of data from the applications, which are displayed in the user interface. With the alarm widgets in mappView, an operator can acknowledge alarms in mappAlarmX.

3.1.2 MappView

To simplify the process of creating HTML5 web pages, B&R developed mappView. Normally, HTML5 web pages are only coded in text format. With mappView, web

pages can be created graphically. Although large sections of mappView have been made for a graphical development environment, there are additional configuration that needs to be written in XML code format.

When the Automation Studio project is compiled and transferred to a B&R device, mappView is stored on a webserver on the Automation Runtime, which is further described in Chapter 2.2.1. MappView can read and write process data to and from Automation Runtime over OPC-UA. These connections between mappView and the Automation Runtime are configured as so-called “bindings”.

Events in mappView are configured as event-bindings. For example, when an operator interacts with a button, or when a monitored process value is updated. When these events occur, suitable actions are subsequently executed.

Within a client’s connection to the webserver, there are session variables and expressions that are session specific to only that client. These session variables can store temporary data of the client’s connection. With the so-called expressions, statements are locally evaluated in the client’s web browser, instead of that evaluation taking place in the Automation Runtime.

An instance of mappView on a webserver can contain one to several different user interfaces. Each of these can be accessed separately by altering the Uniform Resource Locator (URL) in the client web browser. When these user interfaces are configured in Automation Studio, the various contents in a user interface can optionally be set to be pre-cached. If configuring a content to be pre-cached, then upon browsing to the user interface, a load screen appears and the specified contents in the configuration are pre-loaded to memory, before the user interface is shown. Once these contents are pre-loaded to memory, page changes will be quick, since the client device no longer need to request the page content from the webserver. The only disadvantage of pre-caching content is that when the initial pre-loading screen appears, it can take a relatively long time until all the contents have been pre-cached. The loading time will depend on the number of contents that have been configured to be pre-cached. In normal operation

of a powerplant, the user interfaces will always be on, thus the initial pre-caching time delay will only appear upon bootup after a possible power-failure.

The graphics in mappView are vector-based, meaning that the size of the client's screen does not matter, because it can be dynamically scalable. Scalable Vector Graphics (SVG) are preferably used when developing the user interface, although non-vector images can be used as well. These SVG's contain code in XML-format that is interpreted by the client device that displays the graphic. Each element in an SVG (e.g. circles, boxes, lines) has an identification name. In the mappView user interface, properties of SVG elements can be dynamically changed by specifying an ID of an element within an SVG. These elements can be rotated, moved, altered colour, and scaled. When an operator interacts with a user interface containing SVG graphics, the individual selected element by an operator can be distinguished.
MappView has a pre-developed library containing many useful widgets, such as buttons, sliders, static-charts, online-charts, tables, animations, pop-up dialogs and much more.

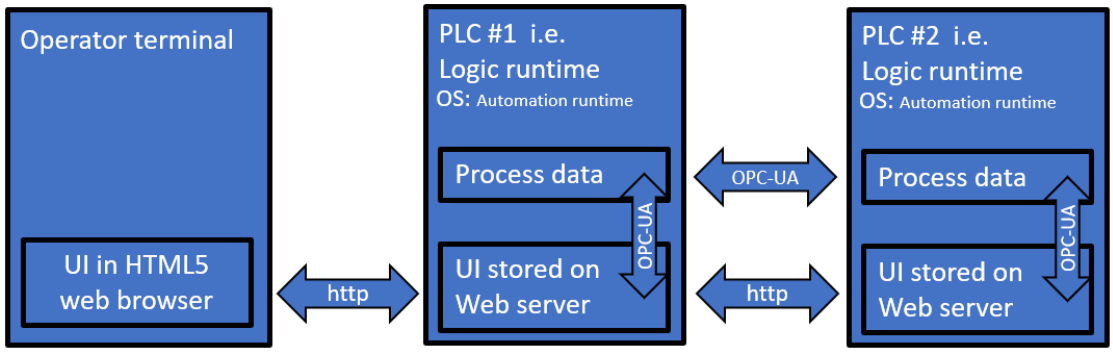

Figure 14 The operator terminal is a client that accesses a user interface stored on a web server on PLC #1. Within the user interface on PLC #1, another user interface is requested from PLC #2.

Within a HTML5 user interface, a window with a web browser can be implemented. Thus, a user interface stored on one PLC's webserver, can access a user interface stored on another PLC's webserver. This is illustrated in Figure 14, where the operator terminal accesses a user interface on PLC #1, who in return subsequently accesses another user interface stored on PLC #2. In this hypothetical scenario, the former

mentioned PLC accesses a smaller user interface only containing a list of alarms, from the latter mentioned PLC. Thus, one user interface can dynamically monitor several other user interfaces. As mentioned in Chapter 2.2.1, the user interfaces are sent over HTTP and process data is sent over OPC-UA.

3.1.3 MappDatabase

Currently, Automation Runtime does not have a DBMS. Thus, if a DBMS is needed, it must be implemented on another OS. However, the mappDatabase application enables the developer to easily configure a connector between the process controller and a database on a general-purpose operating system (GPOS). This connector is currently supported for Microsoft SQL, MySQL, and PostgreSQL. The connector is not limited to these three databases, but the connector is optimized to work with these three databases without any extra hassle. If the chosen database is not one of the three aforementioned databases, additional configuring and coding is required by the developer.

The process controller's operating system Automation Runtime does not directly communicate with the database on the GPOS, instead a python script on the GPOS establishes a communication port to the process controller. Once the python script has established a connection to the process controller, information can be relayed to and from the chosen database on the GPOS. An illustration of this topology can be seen in Figure 15. All user information needed to connect to the database is configured on the process controller's runtime, thus the only configuration for the Python script on the GPOS is a port to communicate with the process controller and additionally specify the IP-address and port-number for the database on the GPOS.

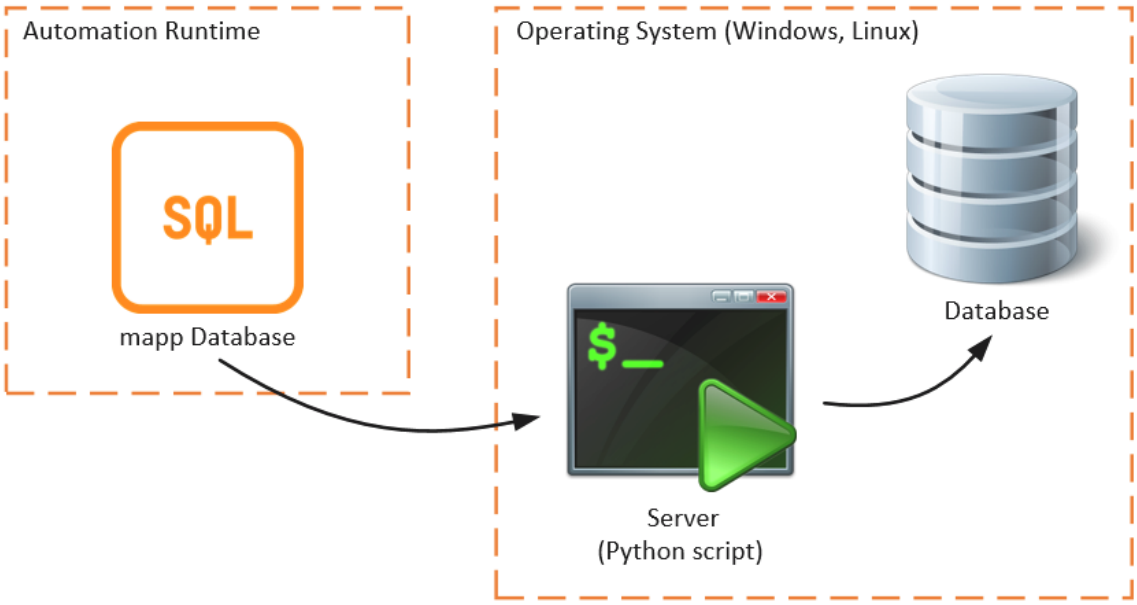

Figure 15 The topology of how the process controller communicates with a database on a GPOS (B&R, 2019).

In the mappDatabase configuration, several types of queries can be specified. There are pre-made structures for the most common SQL-alterations, such as select, insert, update, and delete. Custom queries can be specified, to create custom SQL-commands.

If a structure of a table is created in Automation Studio, that is identical of the table structure in the database, this structure can be specified for a query in the mappDatabase configuration. When data subsequently returns from the database, Automation Runtime can autopopulate the pre-specified table structure, making the configuration of mappDatabase as well as managing queries at runtime quite simplistic for the developer.

In a day to day operation, the process controller could collect process values for a certain time interval and subsequently insert the collected data into the database. A suitable time interval should be chosen. If a longer interval is chosen, a larger temporary data-buffer is required on the process controller. As mentioned in Chapter 2.4.3, although a database is configured for replication, inserts from 50 thousand to 100 thousand rows per second can be achieved. Thus, with a process controller collecting data from several hundred sensors, the time interval is preferably chosen to only be a few seconds.

A temporary buffer of collected data is required, so the database-connector has time to execute other tasks. If an operator chooses to see a chart in the user interface, the process controller can request that data from the database on a GPOS.

An advantage of a time-series DBMS is that a long time-interval with millions of rows of data, can be queried relatively quickly. As an example, a database with a table named MyTable, containing two columns: Datetime and SensorData. The table contain data with a one second sample rate. For example, if a 109-day time interval is queried, there are 9.4 million rows of data in that time interval. If the data is to be plotted in a chart in mappView, a suitable array length should be chosen, depending on the resolution desired. A higher resolution enables the operator to enlarge sections of the chart without a choppy trendline. However, a higher resolution requires more processing power of the client device that the operator uses to request the chart. In this example, the array length is defined to 1744 elements. Thus, when dividing the 9.4 million rows by the array length, one element in the chart equals to 1.5 hours. The 1.5-hour time-slot per returned element, can be specified with the “time_bucket” function in TimescaleDB.

In the customized SQL-code example below, a 109-day interval is specified by giving upper and lower limits for the Datetime. With the time_bucket-function, each returned element is specified to be a 1.5-hour time-slot from the column Datetime. A mean of data in the column SensorData is returned, with the same specified time-slot for Datetime. The Returned data from the database has two columns: QueriedDatetime and QueriedValue, which is ordered by QueriedDatetime ascendingly.

```
SELECT time_bucket('1.5 hours', Datetime) AS QueriedDatetime,
AVG(SensorData) AS QueriedValue
FROM  MyTable
WHERE Datetime > '2019-01-01 12:00:00'
AND   Datetime < '2019-04-20 12:00:00'
GROUP BY QueriedDatetime ORDER BY QueriedDatetime ASC;
```

When the SQL-query above is executed on B&R’s Panel PC 2200, the 1744 rows of averaged data is returned in 11.5 seconds. Thus, when requesting data from the

database, the panel PC 2200 can process 0.8 million rows per second. If expressed in time, 9.4-days' worth of data is returned to the chart every second. By linear extrapolation from this result (which neglects for example effects of possible memory exhaustion), requesting ½ year of data would take roughly 20 seconds.

3.2 Database management system

There are many database management systems available to choose from. As mentioned in Chapter 2.4, time-series databases are optimized for each row of inserted data to be indexed by the current timestamp (e.g. collecting sensor data). Thus, a time-series database is the best choice for collecting sensor data in a powerplant. As mentioned in Chapter 2.4.2, Tallkvist & Warrén (2019) compared the performance of the two time-series databases InfluxDB and TimescaleDB and found that TimescaleDB performed better. Since TimescaleDB is an extension to PostgreSQL, and as mentioned in Chapter 3.1.3, PostgreSQL is supported by mappDatabase, TimescaleDB was chosen as the database for the experiments in this thesis.

Since TimescaleDB is an extension to PostgreSQL, a basic installation of PostgreSQL is needed before TimescaleDB can be installed. Once the extension is installed, tables can be created and subsequently be converted to TimescaleDB's "hyper table". The basic concept of the hyper table is to partition the table into several chunks. A default hyper table is partitioned into 7-day interval chunks, but this interval can be altered at creation or after it has been created. Each chunk can be thought of as a small table within the main table. The main purpose of this partitioning is to enable parallel I/O. This can be beneficial in two scenarios: two different queries can read from the table in parallel or a single query can read from several chunks in parallel.

3.2.1 Streaming replication to a clustered database

In a containerized powerplant, each genset is required to have data stored locally, and if needed, can operate independently from the rest of the powerplant. Thus, the primary database of each genset should be in each respective container.

For redundancy, the database in each genset can be replicated. When a containerized powerplant is sold with several gensets, a common monitoring module is usually sold as well. This common monitoring module would be a suitable location for the replications of the databases. As described in Chapter 2.4.3, there are several types of configurations of replications. In this case, where there are several replications to be stored at one location, a clustered database of replications is a suitable option.

First off, the primary database in each genset, is configured with a time-series database, a replication privileged user, and a replication slot. Once each of the gensets primary database have been configured, the panel PC in the common module can be configured with a clustered database. Since the common module may monitor several gensets, a separate storage medium with large storage and eventual RAID-capabilities would be a good feature. This storage medium could be a Network-Attached Storage (NAS), that would allow all devices connected to the network to access it. A panel PC in the common module can manage the clustered database replicas on the NAS.
Thus, by using the replication slots created on each of the primary databases, the panel PC in the common module can create backups to the NAS. Once the clusters have created backups of the primary databases, each cluster can be registered as a service. Each of the services are configured for hot standby. In hot standby, when changes are made to a primary database, the same changes will apply to the replication server, as also mentioned in Chapter 2.4.3.

3.3 Topology overview

Currently, Wärtsilä's UNIC communicates externally over Modbus. Thus, to receive motor sensor data to the HTML5 user interface, a PLC with Modbus capability is needed. If in the future, external communication from UNIC is configured for OPC-UA, not only is the PLC in the auxiliary module able to communicate with UNIC, but any node in the network with OPC-UA capability will be able to communicate with any of the genset's UNIC. This can be seen in Figure 16, where the engine communicates with the auxiliary module, either first to a PLC or alternatively directly to any Ethernet node in the network, such as a panel PC.

If a panel PC is utilized with hypervisor capabilities, which is further described in Chapter 2.3, the user interface can be stored on an implemented real-time OS on the panel PC. If no panel PC is implemented, or if no real-time OS is implemented on the panel PC, the user interface can be stored on the auxiliary module's PLC. In Figure 16, there are two instances of user interfaces, one is stored on the PLC's webserver and the other one is on the panel PC's real-time OS, where a webserver has been implemented. In this scenario, a web browser on the panel PC's GPOS would be able to access both user interfaces. The user interface on the panel PC's real-time OS is accessed through a virtual port, and the user interface on the PLC over the Ethernet network. In both cases though, HTTP/HTTPS communication is used.

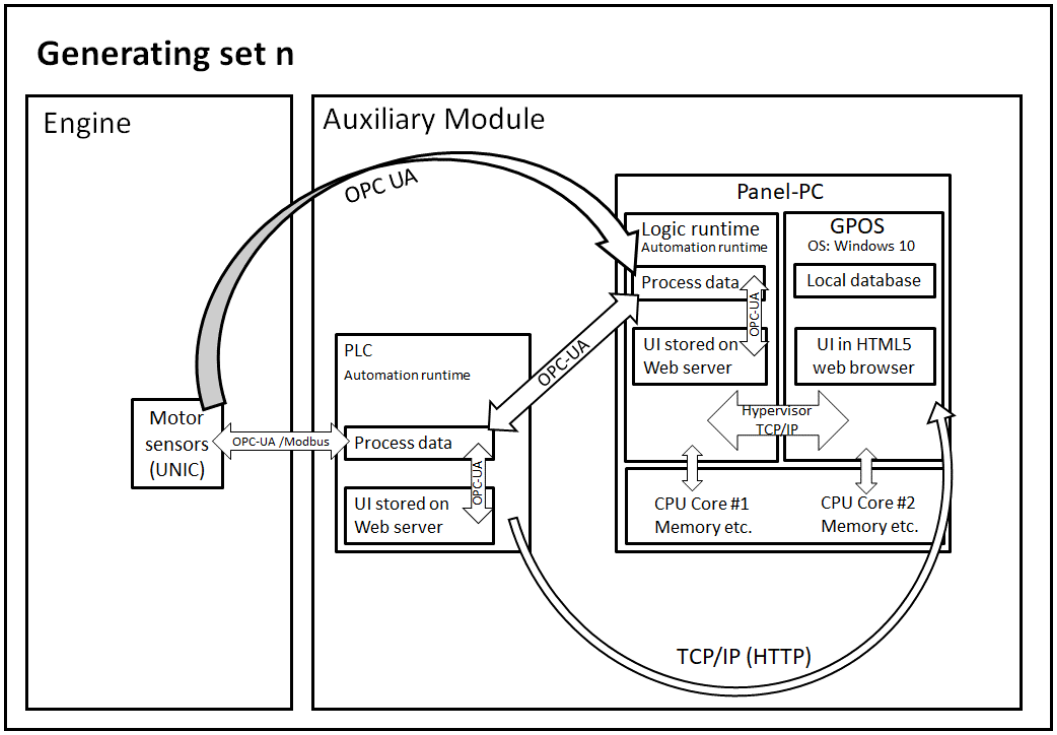

Figure 16 Communication topology in a genset.

Regardless of where the user interface is stored, a GPOS is needed if a database is to be implemented for storage of historical data. In the genset in Figure 16, the database is implemented on the panel PC's GPOS. As previously mentioned, the panel PC's GPOS can communicate with both the PLC's and panel PC's real-time OS's, so the database can communicate with both user interfaces as well. However, if an operator panel is used instead of a panel PC (such as the one in Chapter 4.2), the PLC in the

auxiliary module in Figure 16, must be replaced with an industrial PC with hypervisor capability, where a database can be implemented.

A user interface in the common module should preferably be larger than those in the gensets, since several gensets will be monitored from the user interface in the common module. This could be a panel PC. The process values from the various gensets could be monitored from here. Additionally, the HTML5 user interface pages from each genset can be dynamically interacted with from the panel PC in the common module.

From a redundancy perspective, the genset's panel PC could be implemented as a primary system for the GUI. A backup GUI can be stored on PLC's webserver in the auxiliary module. In case of failure of the panel PC, a quick temporary solution would be to connect any device to the network (such as a laptop) that supports HTML5 web browsing, and from there access the backup GUI. However, if a failure occurs on the panel PC, where a database is implemented, the backup GUI on the PLC will not be able to access the historical data from the panel PC, rather only real-time data can be monitored.

To ensure no loss of data, replication can be implemented for the databases (see Chapter 2.4.3). The primary database could be on each local genset's panel PC and replicas of the databases could be at a common monitoring module, thus ensuring reliability. If the replicas are configured in the common module, the backup user interface on the PLC in the genset's auxiliary module, could be configured to read from the replica database, managed by the common module's panel PC. As mentioned in Chapter 2.4.3, data can only be read from a replica database. Thus, if failure occurs on the genset's panel PC, and the backup GUI on the PLC is used, no data will be logged from that point on, but historical data can be queried up until the time of the failure of the panel PC. Figure 17 illustrates the replication, where a DBMS on the GPOS of the panel PC in the common module is configured to manage several clustered replica databases stored on the NAS, as mentioned in Chapter 3.2.1.

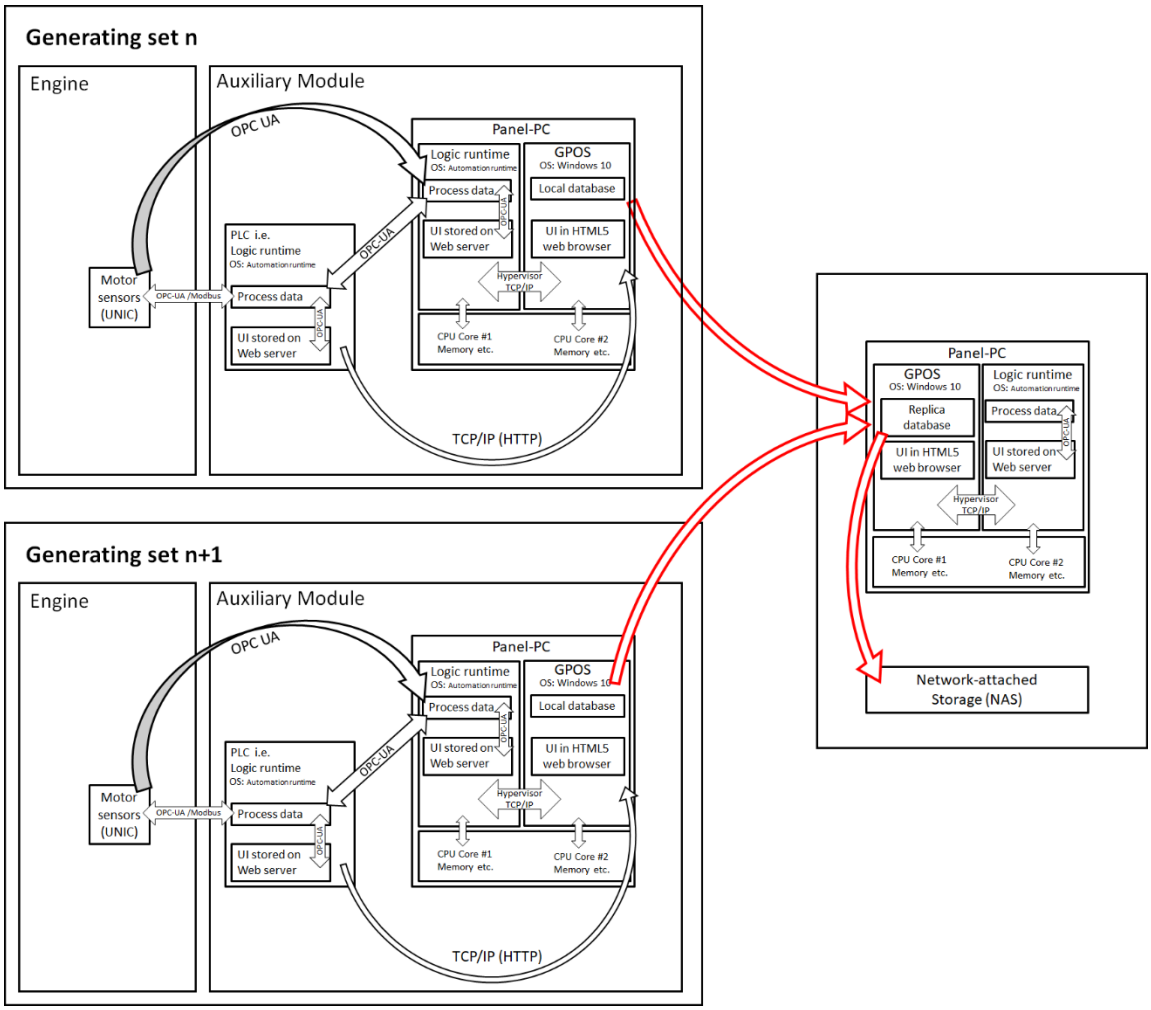

Figure 17 The arrows outlined in red indicate the replication from the gensets to a common module with a NAS.

4 RESULTS OF PERFORMANCE TESTS

User interfaces were created in Automation Studio and subsequently transferred to PLCs, where the user interface source files are on a webserver. Two of B&R's PLCs were used: a X20CP1586 was used to represent a genset, while a X20CP1584 was used to represent the PLC to be implemented in a common module. The user interface can be used by any device in the local network, with a web browser supporting HTML5. A few devices were tested to open the HTML5 user interfaces and interact with them. The results of the performances of these are described in this chapter.

Various functionalities were replicated in the HTML5 user interfaces. The replicated functionalities are used by the current user interface implemented in genset's of the W20 containerized powerplant, as well as in WOIS. To name a few of these functionalities: alarm management, historical charts, online charts, and dynamically manipulating vector graphics. The created user interfaces are not full-fledged user interfaces ready to be used in an actual powerplant. Instead, each of the critical functionalities in the currently used WOIS were replicated, so the performance of these could be evaluated for future possible development.

For historical data storage, a database management system was implemented on a Windows 10 installation. The genset PLC sends real-time data to the database, and if charts of historical data are requested to be seen in the user interface, this data is queried from the database. The database performance is further described in this chapter.

4.1 Simulation with PC

In the development stage of the user interface in Automation Studio, the project can be simulated on the development PC, instead of transferring it over to a real-time operating system (e.g. a PLC). This was useful in the early stages, due to no additional hardware is required to test the project.

Fortunately, at Wärtsilä's offices there were already a bunch of B&R's PLCs for test

purposes. Due to the convenient availability of PLCs, the Automation Studio project could be transferred to a PLC instead of simulating it in the PC. The PLCs were connected to the same LAN-network as the development PC.

Once the user interfaces were developed, they were tested by using a Google Chrome web browser on a laptop. In the web browser address bar, the IP-address of the PLC, along with port number and visualization name could be entered to access the various created user interfaces.

When all the various functions implemented in the user interface were tested with the laptop's web browser, there were no sign of slow response times, despite rapid interaction with the user interface.

4.2 Operator terminal T50

As the first actual industrial touch-panel to be tested, the B&R's 15.6" 6PPT50.156B-16A operator panel was tested. According to the sales personal at B&R, this touch-panel would be a suitable device to be implement to this project. The panel has a simple custom operating system that allows the user to implement it for a selected few tasks. It has two options for user interface: a web browser that support HTML5 and a VNC-viewer (Virtual Network Computing). Additionally, an OPC-UA server can be established on the operator panel. Due to the few possible modes of operation, it is a cost-effective alternative for a user interface.

However, when it comes to the performance of the T50 touch panel, it became clear that the components chosen for the operator panel by the vendor, is meant for a quite light-weight user interface without any advanced functionalities. The device has an ARM Cortex A9 800 MHz dual core processor and 1 GB of RAM.

When the created HTML5 user interface was opened in the T50's web browser, noticeable response delays occurred after interacting with certain functionalities in the user interface. These response delays were long enough to not be acceptable for a user interface in a powerplant. Although many of the simpler functions worked without any

response delays, a few of the more process intensive tasks had response delays.

4.3 Panel PC 2200

After the T50 panel was tested (mentioned in Chapter 4.2), it could be concluded quite quickly that it was not sufficient for the graphics in the created user interface. Thus, the next device to be tested would need better hardware. In B&R's portfolio of operator panels, the T50 panel currently has the best hardware capability. However, if a panel PC is used instead of an operator panel, B&R has several devices with better hardware than the T50 panel.

Thus, B&R's 15.6" panel PC 2200 was ordered as the next device to be tested. This device has an Intel Atom E3940 1.6GHz quad core processor and 8 GB of RAM. As a storage medium, a Compact Flash (CF) card is used. When only comparing the panel PC's hardware to that of the T50 panel, a significant increase of hardware performance is noticeable. However, with more capable components, the panel PC is more expensive than the T50 panel. In relation to the hardware capabilities, the panel PC is a cost-effective device.

The GPOS used on the panel PC was "Windows 10 IoT Enterprise 2016 LTSB". This GPOS is tailored for industrial use, since unlike a regular Windows 10 installation, it will not receive updates as frequently, rather it will only receive security updates. The device has support for hypervisor capability, so on one of the partitions of the panel PC's flash memory, the Automation Runtime can be installed. However, although there was an opportunity of implementing a real-time OS on the panel PC, it was decided that a PLC should be used as the storage medium for the user interface.

The created HTML5 user interface was tested on the panel PC's GPOS. The response delays that occurred while testing the T50 panel (which is mentioned in Chapter 4.2), were not noticeable at all with the panel PC. When the contents in the user interface were set to be pre-cached, pages of the user interface could be changed without response delays, and animations were smooth. All the various functionalities implemented in the user interface operated smoothly.

The various functionalities implemented in the user interface were tested with rapid interaction, to simulate a heavy workload. Meanwhile, the performance of the panel PC was analyzed. The processors utilized roughly 20% of a full work-load. If the contents in the user interface were configured to be pre-cached, there were less peak-performances by the processor, compared to no pre-caching at all. These processor performances can be seen in Figure 18, as a percentage used of the total processing capability. When a chart with a long time-interval was requested from the user interface (i.e. querying data from the database), there was an average processor usage of 33% and peaks up to 50%, as can be seen in Figure 19. The difference of the utilized RAM could not be distinguished by the panel PC, when the contents in the user interface were pre-cached and not pre-cached. The RAM usage of the panel PC was at a constant 15%. When data was requested from the database, the RAM usage temporary increased to 22%. Thus, the panel PC seams to manage the user interface well.

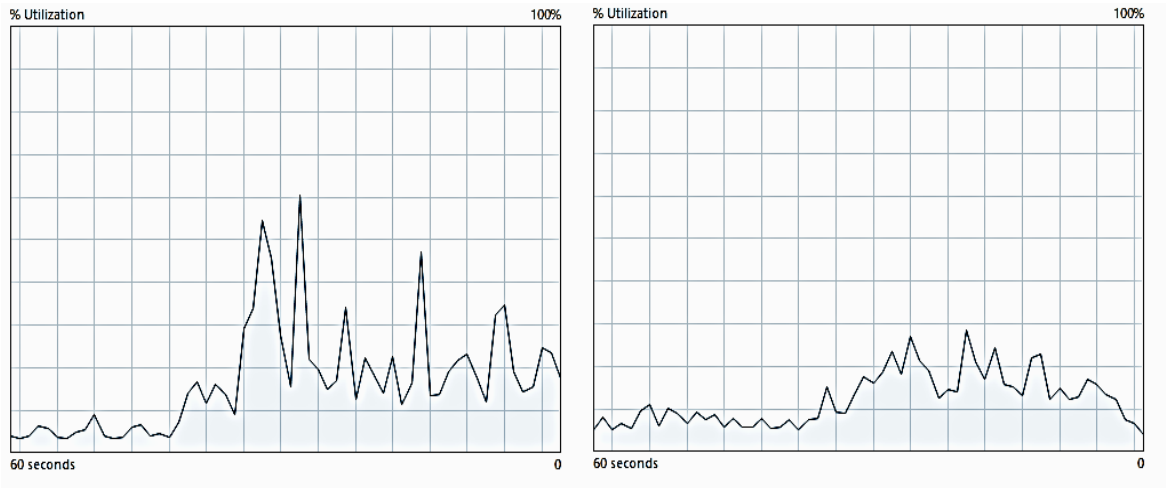

Figure 18 Percentage usage of the panel PC's processor while the user interface is under heavy workload. The picture on the left was with no pre-cache configured, while in the picture on the right-hand side, all contents were pre-cached.

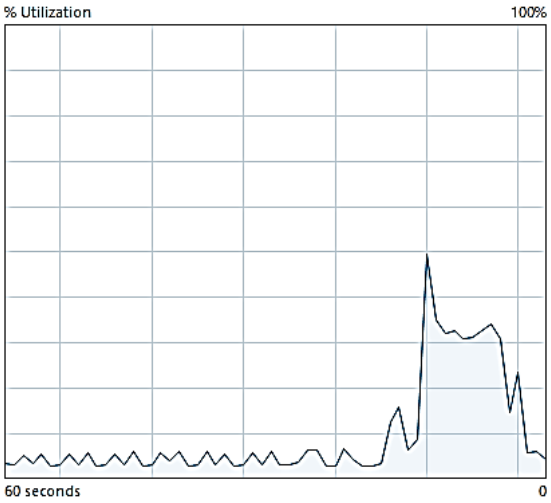

Figure 19 Percentage usage of the panel PC's processor when requested data for a long time-interval from the database on the panel PC.

In the experiments with the panel PC, a Google Chrome web browser was utilized to fetch the HTML5 user interface from the PLCs webserver. In the previous study of web browser performance by Radhakrishnan (2015), mentioned in Chapter 2.2.2, it was found that Google Chrome and Firefox performed best, depending on the tasks. Since the user interface worked very well in Google Chrome, and there were no response delays, Firefox was not evaluated during the experiments in this thesis.

4.4 X20 PLC

The mappView user interface created in Automation Studio must be transferred to an Automation Runtime OS, in order to be utilized. Although the tested panel PC had hypervisor capability, it was decided to use a PLC with Automation Runtime instead. In the auxiliary module of a genset, a PLC is needed regardless of what user interface is used. Thus, this PLC would be the medium where the webserver with user interface would be implemented.

The two B&R's X20 PLCs used in this thesis, has a system diagnostics manager that can be accessed from a web browser of a device connected to the same network. In the system diagnostics manager, several things can be monitored, one of which is the performance of the PLC's CPU usage. While testing the user interface with a heavy workload on the panel PC, mentioned in Chapter 4.3, the performance of the PLC was

monitored simultaneously. The PLC had no other process intensive tasks than the mappView interface, mappAlarmX, and mappDatabase. The PLC's CPU operated on a constant base load of 20% usage of the processor, while momentary peaks jumped to 32%, as can be seen in Figure 20.

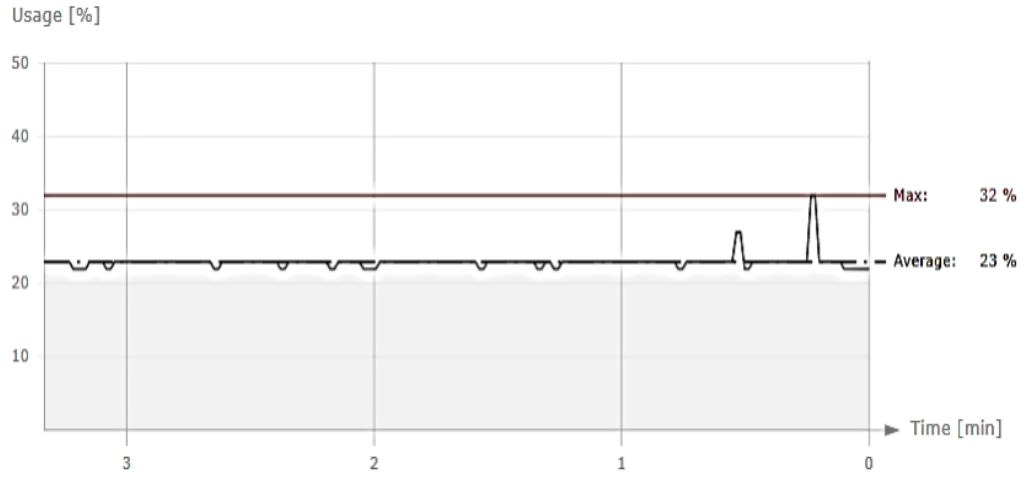

Figure 20 The percentage used of the PLCs processor while the mappView interface opened by a client is under a heavy workload.

4.5 Time-series database

To store historical data from each genset, a DBMS is preferably utilized. In this thesis PostgreSQL with the time-series extension TimescaleDB was chosen, since the application mappDatabase in the utilized PLC has support for PostgreSQL. The database was installed on the GPOS of the panel PC 2200, mentioned in Chapter 4.3.

In tests of the configuration, the PLC collected data with a one second sample rate and subsequently inserted the data into the database periodically. This worked well without any delays or loss of data. The performance of the insert-rate to the database was not further tested, because the data collection sample rate of one second is enough for a powerplant.

In the user interface there are pages with charts of historical data. To display a chart with a specified time-interval, the PLC queries data from the database on the panel PC. In a test, where a 109-day time-interval was requested to be displayed in a chart,

the database on the panel PC was able to process roughly 0.8 million rows of data per second. Since the sample rate of the collected data in the database was one second, 9.4 days' worth of data were returned to the chart every second. By linear extrapolation from this result (which neglects for example effects of possible memory exhaustion), a chart for a half-year time-interval would last roughly 20 seconds to load all data to the chart.

4.5.1 Database replication

To ensure no loss of data, the database can be replicated. In this thesis, this was tested by implementing the primary database on a panel PC, while a replication database was implemented on a laptop. In a powerplant setup, the panel PC in this scenario is in the genset, while the laptop would be a machine in the common module (i.e. a panel PC) that manages the replication database. The panel PC, laptop, and PLC were connected in the same local area network. The PLC inserted collected data into the database on the panel PC. The replication of the database could be witnessed in real-time; when data was inserted into the primary database, the same data could be instantly seen in the replication database.

5 DISCUSSION

The various elements experimented with in this thesis are discussed in the following subchapters.

5.1 Required functionality and performance

The objective of this thesis was to mimic the current server-based WOIS to create a simplified light-WOIS based on HTML5. When interacting with a user interface, there should be no major response delays. In this thesis, the main functionalities from WOIS were realized in the HTML5 user interface.

5.1.1 Touch panel

As mentioned in Chapter 4.2, the tested operator panel did not have enough processing power to display the created HTML5 user interface. Operator panels with HTML5-support in automation vendors portfolios today do not have top tier hardware, rather the hardware has most likely been chosen to be optimized for very simple graphics. The hardware in these devices will most likely be better optimized for more advanced graphics in the future, because the HTML5-based panels are relatively new to the automation industry. If these panels become available in the future, they could be used instead of the panel PC used in this thesis, which has the performance requirement needed. However, if the operator panel is used and a database is needed, the database must be located elsewhere than on the panel PC's GPOS, for example, the PLC in the genset can be replace with a panel PC.

5.1.2 PLC

As mentioned in Chapter 4.4, the CPU performance of the PLC with the webserver was at a constant 20% load, while only configured for running the mappView user interface. The required processing power for the main implemented task on the PLC should primarily be taken into consideration. In the compact powerplant, the PLC in the genset maintains the equipment in the auxiliary module, while the PLC in the

common module monitors each of the gensets.

5.1.3 Compact Flash memory

The tested panel PC in Chapter 4.3 has a Compact Flash memory card as its main storage medium. In the experiments in this thesis, a DBMS was installed on the panel PC's GPOS. Data is collected with a one second sample rate by the PLC and subsequently inserted into the database. The amount of data stored in the database must be limited, when comparing to the server-WOIS storage capability that can have 10 years' worth of data. After discussions at Wärtsilä, the data storage time period was defined to be 6 months' worth of historical data on the database. The storage capacity of Compact Flash memory cards exists up to 512 GB, so the 6 months' worth of data from several hundred datapoints each with a one second sample rate, should not be a problem.

Flash memory cards have a finite amount of program/erase cycles, until the oxide layer of a cell in the memory has degraded to a state that data can no longer be sufficiently retained. In the flash memory's infancy, the lifetime expectancy of the storage medium was not so good. However, with current flash memory technology, the device in which the storage medium is used (e.g. a panel PC) will in most cases malfunction long before the storage medium will malfunction. Thus, although a database is implemented on the Compact Flash memory card and information is written as well as erased frequently, it should not malfunction during the operation of the powerplant.

5.2 Forward compatibility

Implementing one of the touch panels from Chapter 4.2 or 4.3 in a powerplant will ensure that replacement parts will be available long after the powerplant has been commissioned, due to the widely used HTML5 standard.

The communication of process data with OPC-UA, used by the mappView user interface, is widely accepted as a communication protocol that will unify

communication between many vendors. As of writing this thesis, the TSN Ethernet standard is being finalized for OPC-UA by the OPC foundation. B&R, whose hardware has been utilized in this thesis, has announced in press releases that these devices will soon be available for customers. OPC-UA TSN is predicted to be the most commonly used standard in the automation industry, making it interoperable between a wide assortment of devices.

A disadvantage of the mappView user interface, is that it must be stored on a B&R's device with Automation Runtime. Thus, if implemented, mappView relies on B&R's software and that the hardware is available for purchase. If a faulty PLC needs to be replaced after a powerplant has been commissioned, the PLC must be replaced with one of B&R's devices.

5.3 Economical aspect

The price of the server hardware and licensing is very expensive for WOIS. The cost of only the server setup may be ten thousand to tens of thousands of euros. When a powerplant with the server setup is sold to a customer, a PLC in the genset's auxiliary module is needed for the processes there. The price of these PLCs is not taken into consideration in the aforementioned server price.

The hardware utilized in the experiments in this thesis is only a fraction of the server price. The PLCs to be utilized in the genset's auxiliary module can simultaneously run a webserver with the user interface, while the PLC still manages the genset's auxiliary processes. The cost of either an operator panel or a panel PC may be more expensive than an old-fashioned screen with cohering buttons that has been previously utilized in the containerized powerplant genset's auxiliary modules as the user interface. The operator panel in Chapter 4.2 was 678€, while the panel PC in Chapter 4.3 was 1134€. The additional cost for a panel PC instead of a traditional operator panel containing a screen with cohering buttons is neglectable when the reduced total cost is taken into consideration, by not implementing a server.

6 CONCLUSIONS AND RECOMMENDATIONS

The various functionalities used in the current Wärtsilä Operator Interface System (WOIS) were replicated to a light version HTML5-based user interface, by using B&R's Automation Studio with the mappView application. A full-fledged working WOIS was not replicated, rather only its main functionalities. The mappView user interface is transferred and stored on a web server of a B&R's device with its real-time operating system Automation Runtime. In this thesis, mappView was stored on a B&R's X20-series PLC.

The created mappView user interface can be accessed by using any device with a web browser that supports HTML5. Mainly devices connected in the Local Area Network (LAN) can access the webserver, but ports may also be opened to allow an external connection to access the mappView user interface. Two touch screen devices were ordered to be evaluated for using the mappView user interface.

The first touch panel was B&R's 15" T50 operator panel, which has a custom operating system with a VNC-viewer and a web browser. When the mappView user interface was tested with the T50 panel, it could be concluded that some of the implemented functionalities had severe response delays, making the user experience somewhat frustrating. Thus, this panel would be insufficient to be used to operate a light-WOIS.

The second touch panel was B&R's 15" panel PC 2200, on which several operating systems can be installed. The preinstalled GPOS "Windows 10 IoT Enterprise 2016 LTSB" was used in this thesis. The mappView user interface was used with a Google Chrome web browser. When using the mappView user interface on the panel PC and interacting with the various implemented functionalities, there were no occurrences of response delays. Thus, this panel could be an option for light-WOIS.

To store historical data, the DBMS PostgreSQL with the time-series extension TimescaleDB was implemented on the GPOS of the panel PC. The PLC with the Automation Runtime can send and retrieve data from the database with a HTTP

communication. To make the database redundant, the main database was configured to be replicated to another device. In the experiments in this thesis, the replicated database was configured on a laptop, but in an actual powerplant the replica would be a panel PC in the common monitoring module. Regarding read/write of data, and replication of the database, it had a satisfactory performance for use in a powerplant.

The next step with this HTML5-based light WOIS would be to create a full-fledged working user interface. This would determine if implementing many functionalities into a single user interface would decrease the performance of the panel PC 2200 tested in this thesis. If this is proven to be successful, the mappView user interface could further be implemented in a pilot project of either a containerized powerplant or the Wärtsilä modular block.

SWEDISH SUMMARY

Optimerat automationssystem för kompakta motorkraftverk

Inledning

Vanligtvis finns det många automationsprocesser i olika sektioner i en industriell miljö. För att övervaka dessa från ett enda kontrollrum används vanligen SCADA (Supervisory Control and Data Acquisition). SCADA består av serverhyllor i ett flertal serverskåp. Mängden serverskåp beror på hur stor process man övervakar samt för hur lång tid och hur tätt man vill lagra informationen. På dessa servrar finns användargränssnitt samt historiska data. I ett kontrollrum är ett flertal datorer anslutna till SCADA-systemet, som operatörer använder för att övervaka hela industrin. Olika processer i en industri har individuella programmerbara logiska styrenheter som ansvarar för dess lokala process. Alla dessa programmerbara logiska styrenheter samt SCADA är anslutna till ett och samma nätverk. Varje ansluten programmerbar logisk styrenhet i nätverket för dess information vidare till SCADA. Från kontrollrummet är det möjligt att både övervaka processerna och ge kommandon till de individuella programmerbara logiska styrenheterna för att t.ex. ändra önskat processvärde. Detta är en robust metod som är skalbar för en stor industri. Ifall man implementerar SCADA-servermodellen i en mindre industri är det nödvändigtvis inte den mest kostnadseffektiva lösningen, eftersom servrar och licenser kan vara relativt dyra.

I Wärtsiläs W20-containerkraftverk finns en förbränningsmotor med en tillhörande generator i en container medan övrig tilläggsutrustning finns i en annan container bredvid. Detta kallas för ett genereringsset eller genset. Antalet installerade genset bestäms enligt den eleffekt som kunden önskar få. Vid installation av flera genset är det vanligt att även installera en gemensam övervakningsmodul. För att övervaka kraftverket används ett serverbaserat SCADA-system som kallas för Wärtsilä Operator Interface System (WOIS).

Under de senaste åren har automationsleverantörer börjat erbjuda programmerbara logiska styrenheter som det går att lagra HTML5-källfiler på. Som komplement till denna möjlighet har även program utvecklats som ger utvecklaren möjlighet att skapa

HTML5-källfiler med ett semigrafiskt gränssnitt. Jämfört med traditionell utveckling av HTML5-källfiler krävs ingen tidigare kodningskunskap. Därutöver kan man spara in på en hel del utvecklingstid när man använder det semigrafiska gränssnittet istället för traditionell textkodning. Ett användargränssnitt som har stöd för HTML5 kan användas med en enhet med en vanlig webbläsare, som t.ex. en vanlig dator, som är ansluten i samma nätverk som den enhet där källfilerna är lagrade. Automationsleverantörer har även under de senaste åren börjat erbjuda industriella pekskärmar med stöd för HTML5-webbläsare.

Syfte

Vid försäljning av mindre kraftverk är målet att undvika implementeringen av servermodellen av WOIS och istället implementera en mer kostnadseffektiv konfiguration utan att reducera nuvarande funktionalitet av WOIS.

I denna avhandling utvärderades möjligheten att decentralisera nuvarande WOIS, så att det istället finns HTML5-baserade användargränssnitt lagrade på programmerbara logiska styrenheter i olika sektioner av industrin, men som fortfarande kan nås från ett kontrollrum.

Metod

För att skapa det HTML5-baserade användargränssnittet användes B & R:s programvara Automation Studio med applikationen mappView. Applikationen består till en stor del av en grafisk utvecklingsmiljö, men där det ytterligare krävs en del konfigurering i form av kod. Applikationen har ett bibliotek med flera färdigt gjorda gränssnittskomponenter som lätt kan implementeras i användargränssnittet. MappView kräver att det körs på B & R:s realtidssystem Automation Runtime. Realtidssystemet kan implementeras på de flesta av B & R:s produkter som t.ex. en Programmable Logic Controller (PLC) eller en panel-PC. I denna avhandling överfördes och lagrades mappView på en PLC. Ett komplett fungerande användargränssnitt gjordes inte i mappView, men diverse funktioner som finns i dagens WOIS återskapades i HTML5 användargränssnittet.

För att lagra historiska data implementeras vanligen en databashanterare. Realtidssystemet Automation Runtime har ingen databashanterare, så om en databas behövs så bör den installeras på ett annat operativsystem. I denna undersökning implementerades databasen PostgreSQL med tidsserietillägget TimescaleDB på en Windows 10-installation på en panel-PC. Tidsseriedatabaser är lämpade för information som blir lagrad och indexerad enligt tidsstämpel. Informationen kan analyseras gällande hur den förändras över en viss tidsperiod. Processvärden från förbränningsmotorn lagras tillfälligt av en PLC och därefter skickas processvärdena till databasen. För att säkerställa att databasen är redundant konfigurerades den för replikering. En databaskopia konfigurerades på en annan enhet så att ändringar som görs i den primära databasen görs också på kopian.

Resultat och diskussion

Det gjorda HTML5 användargränssnittet testades i första hand i datorns webbläsare, och man kunde konstatera att alla implementerade funktioner fungerade felfritt.

En T50-operatörspanel med en pekskärm beställdes från B & R för att utvärdera om den kan användas för att hämta och interagera med det gjorda HTML5-användargränssnittet. Denna operatörspanel har ganska lätt hårdvara och ett simpelt skräddarsytt operativsystem. Den kan anslutas till en virtuell dator (VNC) och har en HTML5-kompatibel webbläsare. Operatörspanelen drabbades av tidsfördröjningar i samband med vissa funktioner i HTML5-användargränssnittet. Dessa tidsfördröjningar var så långa att användargränssnittet inte kan användas i ett kraftverk.

När prestandan hos den testade operatörspanelen konstaterades vara otillräcklig beställdes en panel-PC från B & R. B & R hade inga operatörspaneler med bättre prestanda i hårdvaran än den som redan testats. Bättre prestanda gick att få i en panel-PC. Panel-PC:n består av en industriell pekskärm och en liten dator inkapslad i samma enhet. Jämfört med operatörspanelen är panel-PC:n 67 % dyrare, men dess prestanda är däremot bättre. På denna panel-PC kan användaren själv välja vilket operativsystem som ska implementeras. I denna undersökning användes operativsystemet Windows 10, men det är också möjligt att implementera realtidssystemet Automation Runtime,

vilket möjliggör att mappView-användargränssnittet kan lagras i panel-PC:n. Det gjorda HTML5-användargränssnittet öppnades och testades i webbläsaren Google Chrome. De tidsfördröjningar som uppstod vid interaktion med operatörspanelen märktes inte när panel-PC:n testades. Alla funktioner som implementerades i användargränssnittet kunde användas utan att någon tidsfördröjning uppstod.

Samtidigt som användargränssnittet användes med Panel-PC:n kunde prestandan analyseras. Panel-PC:s processor var aldrig i full användning och en stor del av arbetsminnet användes inte. Detta tyder på att Panel-PC:n torde klara av att hantera mer processintensiva användargränssnitt än det som testats för denna avhandling. Samtidigt som panel-PC:ns prestanda analyserades analyserades även prestandan hos PLC:n där källfilerna för användargränssnittet lagrats. Cirka 20 % av processorkapaciteten för PLC:n användes, oberoende av vad som testades med användargränssnittet. Att använda processorn för användargränssnittet torde inte påverka kapaciteten att genomföra huvuduppgiften, dvs. att hantera motorns hjälputrustning.

Databasen som installerades på panel-PC:n presterade som förväntat. När data hämtades till en graf i användargränssnittet analyserades 0,8 miljoner rader data i sekunden av panel-PC:n. Kapaciteten att skriva information till databasen testades inte utöver de förhållanden som behövs i kraftverk, vilket i denna studie är bestämd till en samplingshastighet på en sekund. Replikering av databasen testades genom att den primära databasen installerades på panel-PC:n och en kopia av den på en bärbar dator. Information kunde skrivas till den primära databasen och informationen fanns genast därefter även i databasens kopia.

En fortsättningsstudie med HTML5-användargränssnittet vore att göra ett fullt fungerande användargränssnitt som motsvarar nuvarande WOIS med alla dess funktioner, eller åtminstone de flesta. Med detta skulle man kunna avgöra om panel-PC:n fortfarande presterar väl utan några tidsfördröjningar. Om det visar sig att detta fungerar väl, kunde ett pilotprojekt utföras där detta HTML5-användargränssnitt tillämpas. Pilotprojektet kunde utföras antingen på ett W20-containerkraftverk eller på ett Wärtsilä Modular Block.

REFERENCES

B&R, 2018. *Two operating systems on one device.* [Online] Available at: https://www.br-automation.com/sv/om-oss/customer-magazine/2018/20189/two-operating-systems-on-one-device/ [Accessed 16 9 2019].

Bello, L. L. & Steiner, W., 2019. *A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems. Proceedings of the IEEE (Volume: 107 , Issue: 6 , June 2019), Catania, Italy.* DOI: 10.1109/JPROC.2019.2905334

Bloem, J. et al., 2014. *The Fourth Industrial Revolution - Things to Tighten the Link Between it and ot.* [Online] Available at: https://www.sogeti.com/globalassets/global/special/sogeti-things3en.pdf [Accessed 9 9 2019].

Bruckner, D. et al., 2018. *OPC UA TSN - A new Solution for Industrial Communication.* [Online] Available at: https://www.semanticscholar.org/paper/OPC-UA-TSN-A-new-Solution-for-Industrial-Bruckner-Blair/d999e9b223baca245dc84a1da257c40ea4f26961 [Accessed 24 9 2019].

Bruckner, D. et al., 2019. *An Introduction to OPC UA TSN for Industrial Communication Systems.* Proceedings of the IEEE (Volume: 107 , Issue: 6 , June 2019). DOI: 10.1109/JPROC.2018.2888703

Chen, P. M. et al., 1994. *RAID: High-Preformance, Reliable Secondary Storage.* [Online] Available at: https://web.eecs.umich.edu/~pmchen/papers/chen94_1.pdf [Accessed 13 9 2019].

Daneels, A. & Salter, W., 1999. *What is SCADA?.* [Online] Available at: http://cds.cern.ch/record/532624/files/mc1i01.pdf [Accessed 9 9 2019].

Falliere, N., Murchu, L. O. & Chien, E., 2011. *W32.Stuxnet Dossier.* [Online] Available at: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf [Accessed 4 10 2019].

Fatima, H. & Wasnik, K., 2016. *Comparison of SQL, NoSQL and NewSQL databases for internet of things.* 2016 IEEE Bombay Section Symposium (IBSS), Baramati, India. ISBN: 978-1-5090-2730-9. DOI: 10.1109/IBSS.2016.7940198

Gashi, I., Popov, P. & Strigini, L., 2007. *Fault Tolerance via Diversity for Off-the-Shelf Products: A Study with SQL Database Servers. IEEE Transactions on Dependable and Secure Computing (Volume: 4 , Issue: 4 , Oct.-Dec. 2007).* DOI: 10.1109/TDSC.2007.70208

Holloway, M., 2015. *Stuxnet Worm Attack on Iranian Nuclear Facilities.* [Online] Available at: http://large.stanford.edu/courses/2015/ph241/holloway1/ [Accessed 4 10 2019].

Influxdata, 2017. *What is a time series database?.* [Online] Available at: https://www.influxdata.com/time-series-database/

[Accessed 10 9 2019].
Jatana, N. et al., 2012. *A Survey and Comparison of Relational and Non-Relational Database.* [Online] Available at: https://www.ijert.org/research/a-survey-and-comparison-of-relational-and-non-relational-database-IJERTV1IS6024.pdf
[Accessed 9 9 2019].
Jayasamraj, J., 2011. *SCADA Communication & Protocols.* [Online] Available at: https://srldc.in/var/NRC/SRLDC%20Fees%20Feb%202013/Trg%20Psti%20System%20Operator%20Trg/PSO%20PSTI%20%205-17%20Sep-2011%20Adisesha/ppts/Protocols/SCADA%20Communications%20and%20Protocols.pdf
[Accessed 3 10 2019].
KOPČEK, M., 2016. *EMBEDDED PLC WEBSERVER AND POSSIBILITIES OF ITS UTILIZATION.* [Online] Available at: https://www.degruyter.com/downloadpdf/j/rput.2016.24.issue-39/rput-2016-0014/rput-2016-0014.pdf
[Accessed 1 10 2019].
Li, Y. & Manoharan, S., 2013. *A performance comparison of SQL and NoSQL databases. 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Victoria, BC, Canada. ISBN: 978-1-4799-1501-9. DOI: 10.1109/PACRIM.2013.6625441*
Mahnke, W. & Leitner, S.-H., 2009. *OPC Unified Architecture - The future standard for communication and information modeling in automation.* [Online] Available at: https://library.e.abb.com/public/75d70c47268d78bfc125762d00481f78/56-61%203M903_ENG72dpi.pdf
[Accessed 18 9 2019].
Microsoft, 2018. *Time series solutions.* [Online] Available at: https://docs.microsoft.com/en-us/azure/architecture/data-guide/scenarios/time-series
[Accessed 10 9 2019].
OPC Foundation, 2017. *Unified Architecture: Part 1: Overview and Concepts.* [Online]
Available at: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts/
[Accessed 19 9 2019].
OPC Foundation, 2018. *Unified Architecture: Part 12: Discovery and Global Services.* [Online]
Available at: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-12-discovery-and-global-services/
[Accessed 23 9 2019].
PostgreSQL, 2019. *Creating a Database Cluster.* [Online] Available at: https://www.postgresql.org/docs/current/creating-cluster.html
[Accessed 29 10 2019].

Radhakrishnan, J., 2015. *Hardware dependency and performance of JavaScript engines used in popular browsers. 2015 International Conference on Control Communication & Computing India (ICCC), Trivandrum, India. ISBN: 978-1-4673-7349-4.* DOI: 10.1109/ICCC.2015.7432981

severalnines, 2018. *An Overview of Logical Replication in PostgreSQL.* [Online] Available at: https://severalnines.com/database-blog/overview-logical-replication-postgresql [Accessed 29 10 2019].

Tallkvist, D. & Warrén, L., 2019. *Time Series databaser för sensorsystem.* [Online] Available at: http://www.diva-portal.org/smash/get/diva2:1317787/FULLTEXT01.pdf [Accessed 13 9 2019].

Taylor, C., 2018. *Guide to Hypervisors.* [Online] Available at: https://www.serverwatch.com/server-trends/guide-to-hypervisors.html [Accessed 30 10 2019].

TimescaleDB, 2018. *High availability and scalable reads in PostgreSQL.* [Online] Available at: https://blog.timescale.com/blog/scalable-postgresql-high-availability-read-scalability-streaming-replication-fb95023e2af/ [Accessed 13 9 2019].

van der Veen, J. S., van der Waaij, B. & Meijer, R. J., 2012. *Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual. 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA. ISBN: 978-1-4673-2892-0.* DOI: 10.1109/CLOUD.2012.18

Wärtsilä, 2019. *WÄRTSILÄ MODULAR BLOCK.* [Online] Available at: https://www.wartsila.com/energy/modular-block [Accessed 10 2019].

Wassilew, S. et al., 2016. *Transformation of the NAMUR MTP to OPC UA to allow Plug and Produce for Modular Process automation. 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA),* Berlin, Germany. ISBN: 978-1-5090-1314-2. DOI: 10.1109/ETFA.2016.7733749